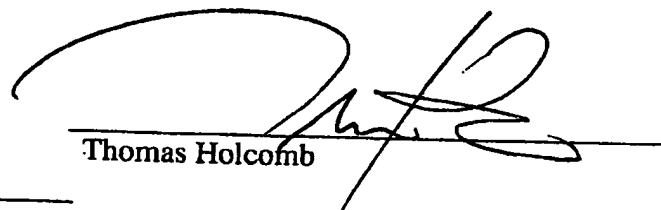# DECLARATION OF THOMAS HOLCOMB

I declare:

1. I have been an employee of Microsoft Corporation since 1999.

2. In 2002, I was a Software Development Engineer in the codec group of the Digital Media Division at Microsoft Corporation.

3. In 2002, I was the primary author of the Windows Media Video V9 Decoding Specification ("WMV9 Decoding Specification"). Other persons at Microsoft Corporation edited and updated the WMV9 Decoding Specification in 2002 and 2003.

4. I have searched my records, searched records available to me, and consulted with persons I believe to be knowledgeable about archived versions of the WMV9 Decoding Specification, so as to find archived versions of the WMV9 Decoding Specification from 2002.

5. On information and belief, among archived versions of the WMV9 Decoding Specification, the version attached as Exhibit A is the most recent version of the WMV9 Decoding Specification from before September 4, 2002.

6. On information and belief, the version of the WMV9 Decoding Specification attached as Exhibit A was last modified August 6, 2002. *See* Exhibit B.

The undersigned declares that all statements made herein of his knowledge are true and that all statements made on information and belief are believed to be true and further, that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment under Section 1001 of Title 18 of the United States Code, and that any such willful false statements may jeopardize the validity of a patent application or any patent issuing thereon.

Executed at the place and date opposite the signature below.

Thomas Holcomb

At ___Redmond, WA___
        (City and State)
on this _12th_ day of March, 2008.

Digital Media Division

WMV Version 9.0

Windows Media Video V9
Decoding Specification

Exhibit A

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This document defines the bitstream syntax and semantics for compressed video data in Microsoft WMV9 format. It also describes the complete process required to decode the bitstream and gives a brief overview of the compression algorithm.

1.1 Notation

The following notation is used in this document.

1.1.1 Arithmetic Operators

+	Addition.
−	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
− −	Decrement.
*	Multiplication.
^	Power.
/	Integer division with truncation towards zero. For example, 7/4 and −7/−4 are truncated to 1 and −7/4 and 7/−4 are truncated to −1.
//	Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and -3//2 is rounded to -2.
##	Rest of the line is a comment.
| |	Absolute value.
	$\mid x \mid = x$, when $x > 0$
	$\mid x \mid = 0$, when $x == 0$
	$\mid x \mid = -x$, when $x < 0$
%	Modulus operator. Defined only for positive numbers.
Sign( )	Sign.
	$Sign(x) = 1$, when $x >= 0$
	$Sign(x) = -1$, when $x < 0$
INT ( )	Truncation to integer operator. Returns the integer part of the real-valued argument.
NINT ( )	Nearest integer operator. Returns the nearest integer value to the real-valued argument. Half-integer values are rounded away from zero.
sin	Sine.
cos	Cosine.
max	Maximum of the arguments.
min	Minimum of the arguments.
exp	Exponential.
$\sqrt{}$	Square root.
$\log_2$	Logarithm to base 2.
median3 ()	Median of 3 values (see section 1.1.8 for definition)

median4 ()   Median of 4 values (see section 1.1.8 for definition)

1.1.2 Logical operators

|| Logical OR.

&&   Logical AND.

!   Logical NOT

1.1.3 Relational operators

>   Greater than.

>=   Greater than or equal to.

<   Less than.

<=   Less than or equal to.

==   Equal to.

!=   Not equal to.

1.1.4 Bitwise operators

A twos complement number representation is assumed where the bitwise operators are used.

&   AND

| OR

>>   Shift right with sign extension.

<<   Shift left with zero fill.

1.1.5 Assignment

=   Assignment operator.

1.1.6 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bit stream.

uimsbf   Unsigned integer, most significant bit first.

vlclbf   Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written.

1.1.7 Bitstream Parsing Operations

The pseudo-code examples use the following bitstream parsing operations

get_bits(n)   Reads n bits from the bitstream and returns the value.

vlc_decode()   Decodes the next variable-length codeword in the bitstream and returns the decoded symbol

1.1.8 Definition of Median3 and Median4

The functions median3() and median4() are used in some of the pseudocode examples in this spec. The functions median3 and median4 are computed as illustrated in the following pseudocode examples.

```
median3 (a, b, c)
{
```

```
    if (a > b)
    {
        if (b > c)
            median = b
        else if (a > c)
            median = c
        else
            median = a
    }
    else if (a > c)
        median = a
    else if (b > c)
        median = c
    else
        median = b

    return median
}

median4 (a, b, c, d)
{

    max = min = a

    if (b > max)
        max = b
    else if (b < min)
        min = b

    if (c > max)
        max = c
    else if (c < min)
        min = c

    if (d > max)
        max = d
    else if (d < min)
        min = d

    median =  (a + b + c + d - max - min) / 2

    return median
}
```

1.2 Decoder Limitations

1.2.1 Minimum and maximum frame sizes

The decoder must support a minimum frame width of 8 samples and a minimum height of 8 lines. The maximum frame dimensions are not limited.

1.3 Memory Requirements

Memory requirements for the decoder are implementation dependent. The following sections describe the memory usage for Microsoft's core WMV9 decoder running on an X86 processor.

1.3.1 Code Memory Size

The executable code for the core decoder module requires approximately 410 Kbytes of memory.

1.3.2 Initialized Data Memory Size

The initialized data used by the decoder is approximately 160 Kbytes.

1.3.3 Dynamic Memory Size

The decoder dynamically allocates memory for frame buffers and other structures. Following are the size requirements.

For the frame buffer, the following formula indicates the memory requirement:

$frame\_memory = \{(h + 64) \times (w + 64) + [(h / 2 + 32) \times (w / 2 + 32)] \times 2\}$ bytes

where:

w = the width of the frame (in pixels) rounded up to the nearest multiple of 16

h = the height of the frame (in pixels) rounded up to the nearest multiple of 16

The Microsoft WMV9 decoder allocates memory for four frame buffers: one for the reconstructed frame, one for the reference frame and two for postfilter processing. Therefore, the total memory allocated for frame buffers is 4 x $frame\_memory$.

1.4 New Features after porting kit's Beta2 release.

- Added B frame for both Progressive and Interlace modes.
- 4x4 transform for P frames.
- Added range reduction scheme.
- Overlap transform.
- Make 16-bit new transform the unique transform for Simple and Main profile.
- Added adaptable inverse quantization scheme.
- Fractional QP for HBR.
- Changed filter design in multi_resolution filter.
- Switched UV MC filter from bicubic filter to bilinear filter.
- Turned rounding control back on.
- Removed frame-to-frame dependency in frame header's Dquant coding
- Added loopfilter for 4x4 DCT boundaries.
- Added start code.

Bitstream related bug fix.
- Unify 4MV's MV pulled back.

2 Source Coder

2.1 Progressive Coding Mode

2.1.1 Input Format

The input format is YUV 4:2:0. Figure 1 shows the spatial relationship between the luma and chroma samples.
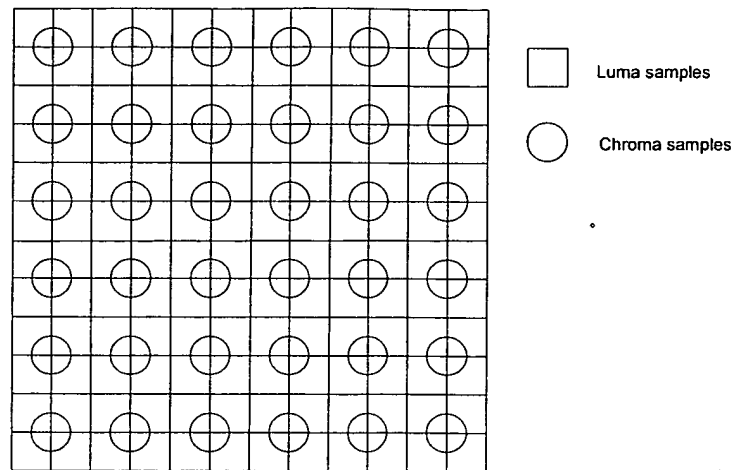
Figure 1: 4:2:0 Luma and chroma sample positions

2.1.2 Hierarchical Elements

The WMV codec decomposes each frame into three hierarchical layers – picture, macroblock and block. Figure 2 illustrates the three layers.
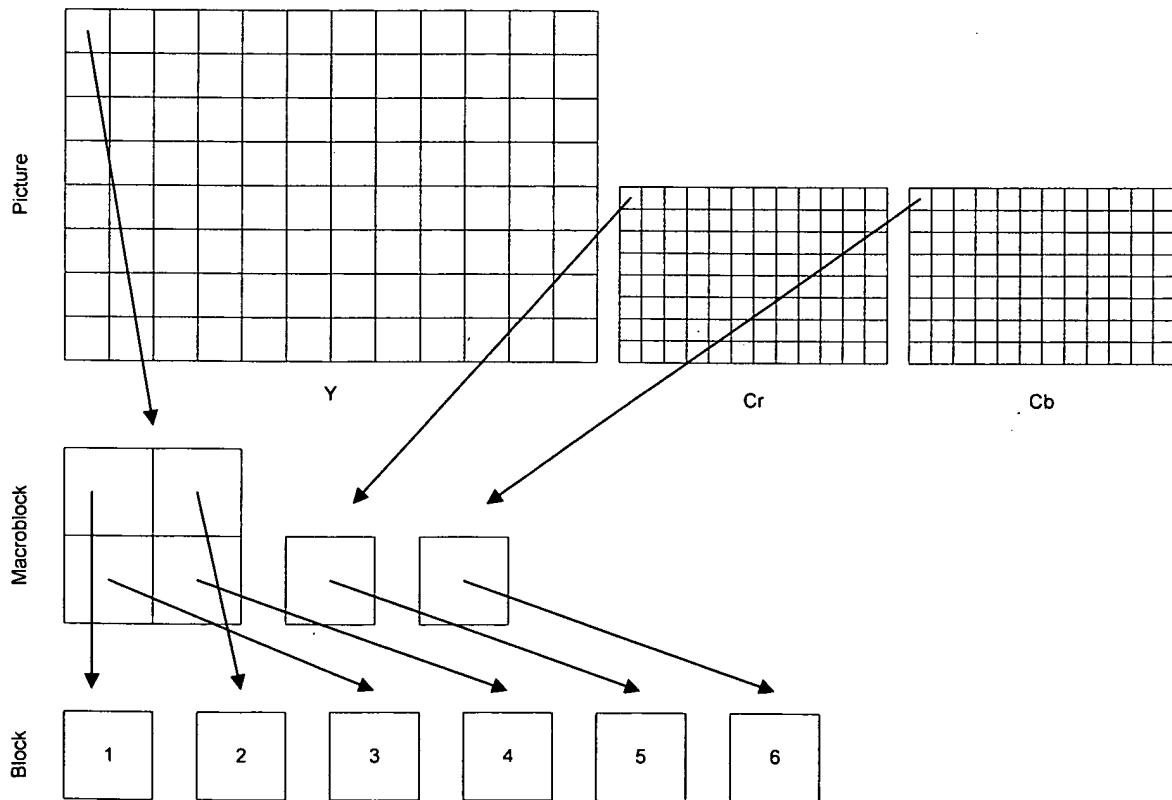
Figure 2: Coding Hierarchy

2.1.3 Coding Description

The compression process uses block-based motion predictive coding to reduce temporal redundancy and transform coding to reduce spatial redundancy. Figure 3 and Figure 4 illustrate the basic steps used to compress the video data in the WMV9 compression algorithm.

Predicted block
(Quantized DCT)

DC Prediction

DC VLE

8x8 pixel block

DCT

8x8 DCT Coeffs

Quantize

AC Prediction

Quantized DCT

run level last

AC VLE

RLE

.....

zig-zag scan
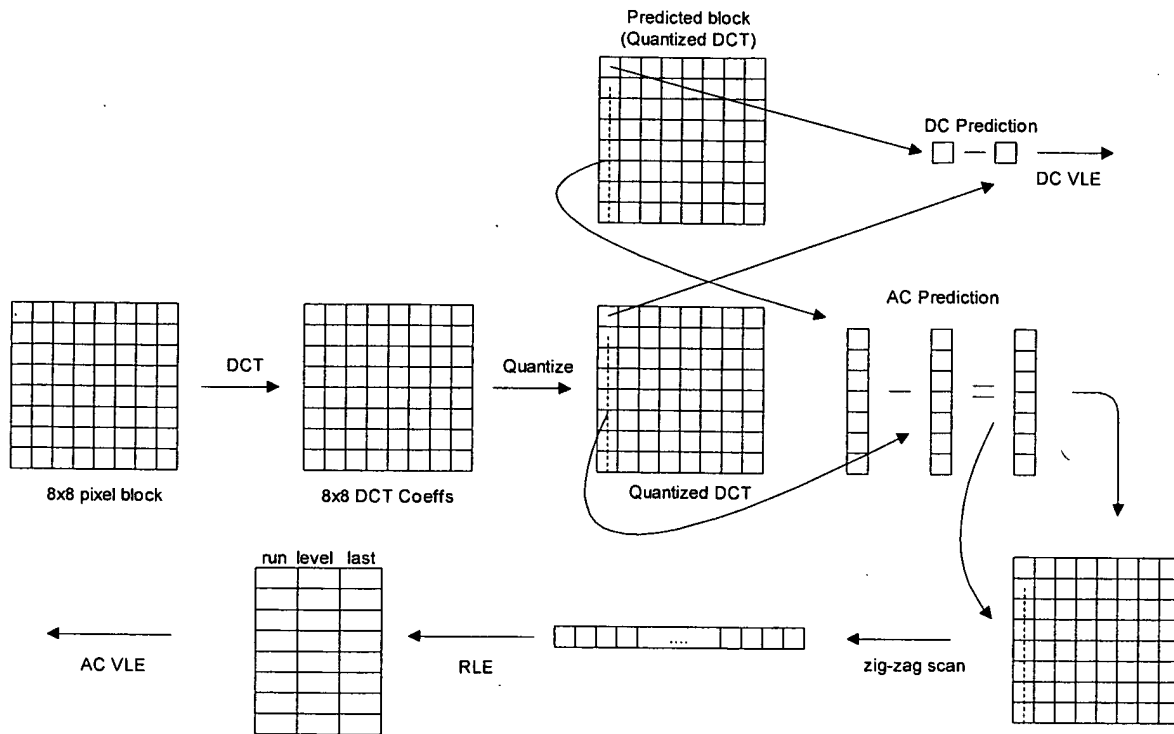
Figure 3: Coding of Intra blocks

Current 8x8 block

Predicted 8x8
block

Motion
Estimation

MVx, MVy

Motion vector
(specifies
displacement of
predicted block in
reference frame)

Error block

Run Level Last

Top 8x4 Error
block

DCT/
Quantize

Zig-zag
scan

....

RLE

VLE

Run Level Last

Bottom 8x4
Error block

DCT/
Quantize

Zig-zag
scan

....

RLE

VLE
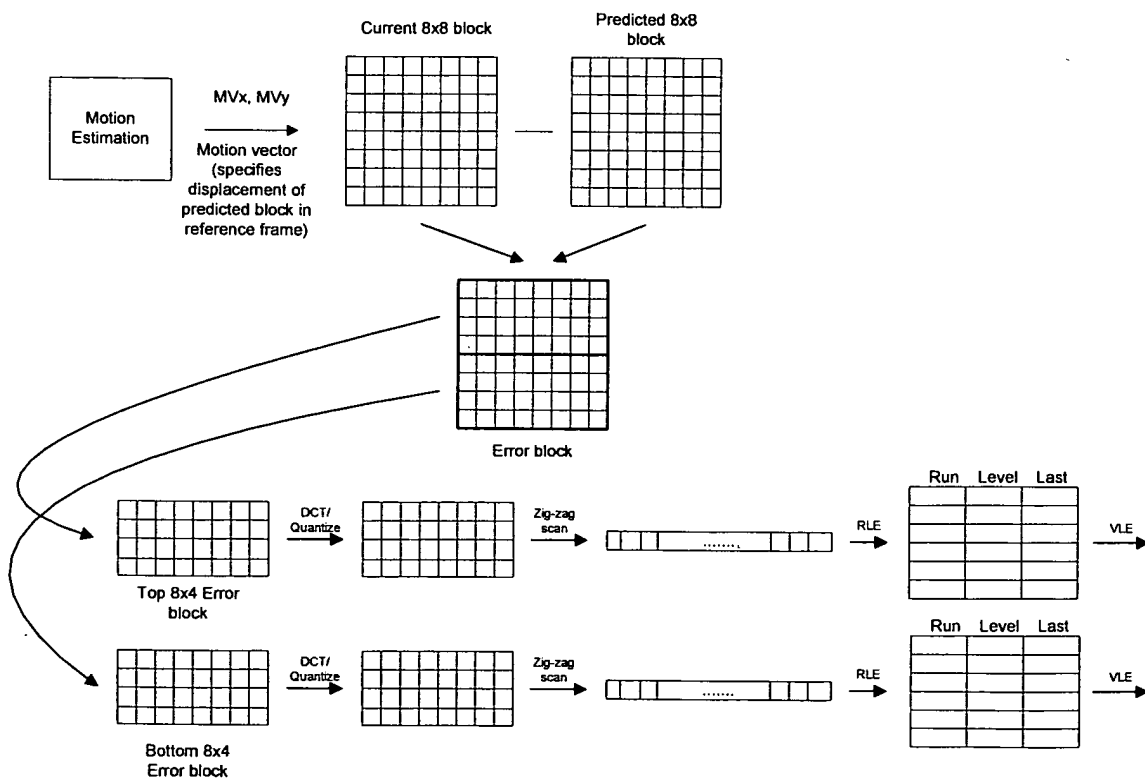
Figure 4: Coding of Inter blocks

2.2 Interlace Coding Mode

2.2.1 Input Format

The input format is YUV 4:1:1. Figure 5 shows the spatial relationship between the luma and chroma samples.

Luma samples
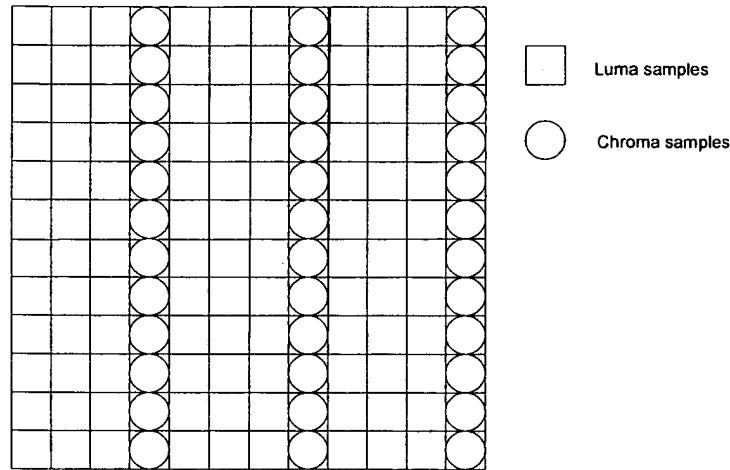
Chroma samples

Figure 5: 4:1:1 Luma and chroma sample positions

2.2.2 Hierarchical Elements

The WMV codec decomposes each frame into three hierarchical layers – picture, macroblocks (MB), and blocks. A picture consists of one whole frame of video, which is then subdivided into macroblocks. Each macroblock consists of 16 by 16 luminance (Y) pixels and two 4x16 chrominace (U and V) pixels. Each macroblock will be subdivided and arranged into blocks where there are four 8x8 Y blocks (Y_0, Y_1, Y_2, Y_3) and two 4x8 U (U_0, U_1) blocks and two 4x8 V (V_0, V_1) blocks.

There are two types of macroblock, frame macroblock or field macroblock. The difference between the two is how the data are arranged to form the blocks Y_i's, U_i's and V_i's. For a frame MB, the Y_i's U_i's and V_i's are formed by simply subdividing the MB as shown in Figure 6.
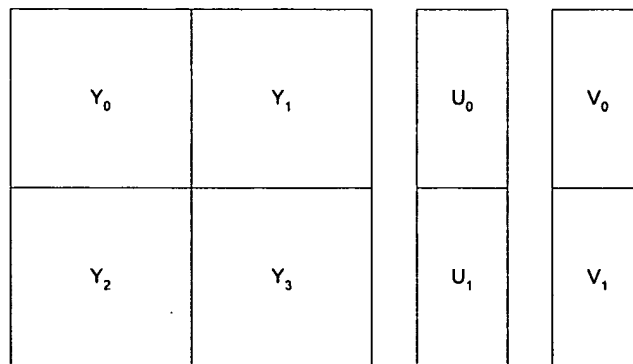
Figure 6: Definition of Blocks in Frame MB in Interlace Coding Mode

On the other hand, the blocks are formed differently for a field MB. In this case, we first permute the macroblock line by line so that all the top fields are placed consecutively followed by the bottom fields. More specifically, we arrange the macroblock so that lines 1, 3, 5, ..., 15 comes first and then followed by lines 2, 4, ..., 16 for Y, U, and V components. Then we subdivide the permuted MB into blocks as shown in Figure 6. Therefore, blocks Y_0, Y_1, U_0, V_0 contains only the top field and blocks Y_2, Y_3, U_1, V_1 contains only the bottom field.

For a frame MB, we typically arrange and send data by color. This means that we send the luminance blocks then followed with the chrominance blocks (i.e., Y_0, Y_1, Y_2, Y_3, U_0, U_1, U_0, V_1).

For a field MB, we typically arrange and send data by fields. This means that we send the top field data, followed by the bottom field data (i.e. Y_0, Y_1, U_0, V_0, Y_2, Y_3, U_1, V_1).

3 Syntax and Semantics

The syntax diagrams are shown in Figure 7 through Figure 25. A guide for interpretation of the diagrams consists of the following:

1. Arrow paths show the possible flows of syntax elements. Any syntax element which has zero length is considered absent for arrow path diagramming

2. Abbreviations and semantics for each syntax element are as defined in later clauses.

3. Syntax element fields shown with square-edged boundaries indicate fixed-length fields those with rounded boundaries indicate variable-length fields and those with a rounded boundary within an outer rounded boundary indicate a field made up of simpler syntax elements which are elaborated on in another section.

4. A fixed-length field is defined to be a field for which the length of the field is not dependent on the data in the content of the field itself. The length of this field is either always the same, or is determined by the prior data in the syntax flow.

5. The term "layer" is used to refer to any part of the syntax that can be understood and diagrammed as a distinct entity. The next-lower layer element in each layer diagram is indicated by a rectangle within a rectangle.

Unless specified otherwise, the most significant bit is transmitted first. This is bit 1 and is the left most bit in the code tables in this Recommendation. Unless specified otherwise, all unused or spare bits are set to "0".

Since the following sections describe the contents of the video bitstream it is often convenient to denote the elements in binary representation. To avoid confusion with decimal representation, whenever a number is expressed in binary format, it is enclosed in square brackets.

3.1 Sequence-level Syntax and Semantics

A header is embedded in the ASF file that contains sequence-level parameters used to decode the sequence of compressed frames. Figure 7 shows the bitstream elements that make up the sequence layer.
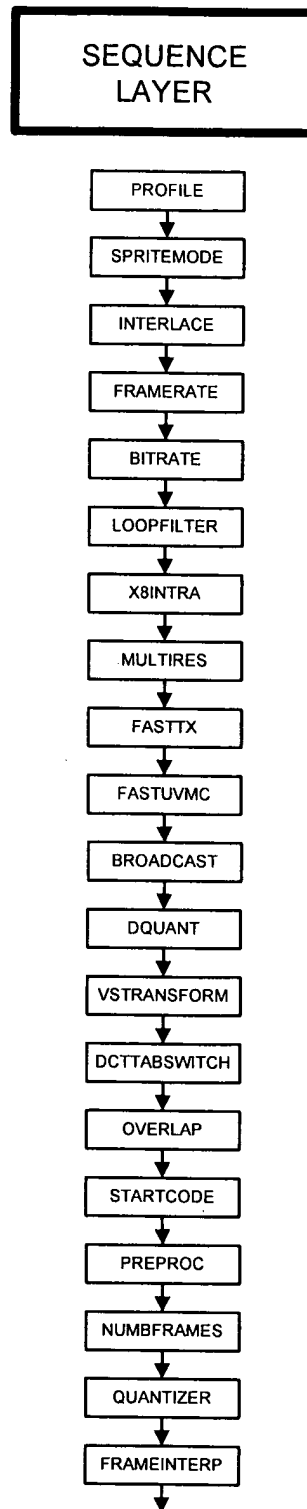
Figure 7: Syntax diagram for the sequence layer bitstream

3.1.1 Clip Profile (PROFILE)(2 bit)

PROFILE is a 2-bit field that specifies the encoding profile used to produced the clip. The three profiles are simple, main, and complex profile and they correspond to PROFILE = 0, 1, and 2, respectively. The simple profile is designed to ease the computation load for the codec by placing restrictions on certain compression tools. More specifically, the simple profile disables X8 Intraframe, Loop filter, DQuant, and Multires while enabling the Fast Transform. The main profile is has all the simple profile tools plus loop filter, dquant, and multires. The complex profile has X8 Intraframe and uses the normal idct transform.

3.1.2 Sprite Mode (SPRITEMODE)(1 bit)

SPRITEMODE is a 1-bit field that specifies whether sprite mode is used to decode the frames. If SPRITEMODE = 1 then sprite mode is used, otherwise it is not.

3.1.3 Interlace (INTERLACE)(1 bit)

INTERLACE is a 1-bit field that specifies whether the video data is coded in progressive or interpaced mode. If INTERLACE = 0 then the video frames are coded in progressive mode. If INTERLACE = 1 then the video frames are coded in interlaced mode.

3.1.4 Frame Rate (FRAMERATE)(3 bits)

FRAMERATE is a 3-bit field that specifies the target number of frames per second in the encoded bitstream. The frame rate is encoded as (FRAMERATE * 4 + 2) frames/second.

3.1.5 Bit Rate (BITRATE)(5 bits)

BITRATE is a 6-bit field that specifies the target bitrate for the video bitstream in kilobits per second (kbps). If the target bitrate is higher than can be expressed in this field (greater than 2047 kbps) then this field contains all ones, signaling that the bitrate is greater than 2047 kbps. The bitrate is encoded as (BITRATE * 64 + 32) kbps.

3.1.6 Loop Filter (LOOPFILTER)(1 bit)

LOOPFILTER is a 1-bit field that indicates whether loop filtering is enabled for the sequence. If LOOPFILTER = 0 then loop filtering is not enabled. If LOOPFILTER = 1 then loop filtering is enabled. See section 4.6 for a description of loop filtering.

3.1.7 X8INTRA I Picture Coding (X8INTRA)(1 bit)

X8INTRA I picture coding is permanently disabled for WMV9 Simple and Main Profiles, regardless of the X8INTRA bit. It is enabled for the complex profile. See section 4.3 for a description of X8INTRA I picture coding.

3.1.8 Multiresolution Coding (MULTIRES)(1 bit)

MULTIRES is a 1-bit field that indicates whether the frames can be coded at smaller resolutions than the specified frame resolution. If MULTIRES = 1 then a frame level field is present which indicates the resolution for that frame. See sections 4.1.1.5 and 4.3.1 for a description of multiresolution decoding in I pictures and 4.3.3.2 for a description of multiresolution decoding in P pictures.

3.1.9 Fast Transform (FASTTX)(1 bit)

FASTTX is a 1-bit field that indicates whether the fast transform is used for the encoding/decoding. If FASTTX = 1, then the fast transform is used in place of the normal IDCT's (8x8, 4x8, 8x4, 4x4). The simple and main profile always use the fast transform. The complex profile uses the normal IDCT's.

3.1.10 FAST UV Motion Comp (FASTUVMC)(1 bit)

FASTUVMC is a 1-bit field that controls the subpixel interpolation and rounding of chroma motion vectors. If FASTUVMC = 1 then the chroma motion vectors that are at quarter pel offsets will be rounded to the nearest full pel positions and bilinear filtering will be used for all chroma interpolation (see section 4.4.4.2). If FASTUVMC = 0 then no special rounding or filtering is done for chroma. The purpose of this mode is speed optimization of the decoder.

FASTUVMC is always 1 for the Simple Profile.

3.1.11 Broadcast Mode (BROADCAST)(1 bit)

BROADCAST is a 1-bit field that indicates whether the Broadcast mode is turned on or off. The Broadcast mode is turned off for the Simple Profile, regardless of this bit. For the Main Profile, the Broadcast mode indicates the possiblility of extended motion vectors in P frames

3.1.12 Macroblock Quantization (DQUANT)(2 bit)

DQUANT is a 2-bit field that indicates whether or not the quantization step size can vary within a frame. There are three possible values for DQUANT. If DQUANT = 0, then the only one quantization step size (i.e. the frame quantization step size) can be used per frame. If DQUANT = 1 or 2, then we allow the possibility to quantize each macroblocks in the frame differently. More detail will be described in section 3.2.1.27.

3.1.13 Variable Sized Transform (VSTRANSFORM)(1 bit)

VSTRANSFORM is a 1-bit field that indicates whether variable-sized transform coding is enabled for the sequence. If VSTRANSFORM = 0 then variable-sized transform coding is not enabled. If VSTRANSFORM = 1 then variable-sized transform coding is enabled. See section 4.4.5.2 for a description of variable-sized transform coding.

3.1.14 DCT Table Switching (DCTTABSWITCH)(1 bit)

DCTTABSWITCH is a 1-bit field that indicates whether DCT coding set switching is enabled for the sequence. If DCTTABSWITCH = 0 then DCT coding set switching is not enabled. If DCTTABSWITCH = 1 then DCT coding set switching is enabled. See section 4.1.3.4 for a description of DCT coding set switching.

3.1.15 Overlapped DCT Flag (OVERLAP) (1 bit)

OVERLAP is a 1-bit flag that indicates whether overlapped DCTs are used. If OVERLAP = 1 then overlapped DCT is used, otherwise they are not used.

3.1.16 Startcode flag (STARTCODE) (1 bit)

STARTCODE is a 1-bit flag that indicates whether startcode synchronization markers are present in the bitstream. If STARTCODE =1 then the markers are present, otherwise they are not present.

3.1.17 Preprocessing Flag (PREPROC) (1 bit)

PREPROC is a 1-bit field that indicates whether preprocessing is used for each frame. If PREPROC = 1 then there is a field in each frame header (PREPROCFRM) that indicates whether preprocessing is used for that frame. If PREPROC = 0 then preprocessing is not used. PREPROC is only supported for progressive encoding.

3.1.18 Number of consecutive B frames (NUMBFRAMES) (3 bits)

NUMBFRAMES is a 3-bit field that indicates the number of consecutive B frames between I or P frames. If NUMBFRAMES = 0 then there are no B frames in the sequence.

3.1.19 Quantizer Specifier (QUANTIZER) (2 bits)

QUANTIZER is a 2-bit field that indicates the quantizer used for the sequence. The quantizer types are encoded according to Table 1.

Table 1: Quantizer specification

FLC	Quantizer specification
00	Quantizer implicitly specified at frame level
01	Quantizer explicitly specified at frame level
10	5 QP deadzone quantizer used for all frames
11	3 QP deadzone quantizer used for all frames

3.2 Picture-level Syntax and Semantics

Each compressed video frame is made up of data structured into three hierarchical layers. This section describes the syntax and semantics of these layers. From top to bottom the layers are:

- Picture
- Macroblock
- Block

Figure 8 through Figure 25 show the bitstream elements that make up each layer.
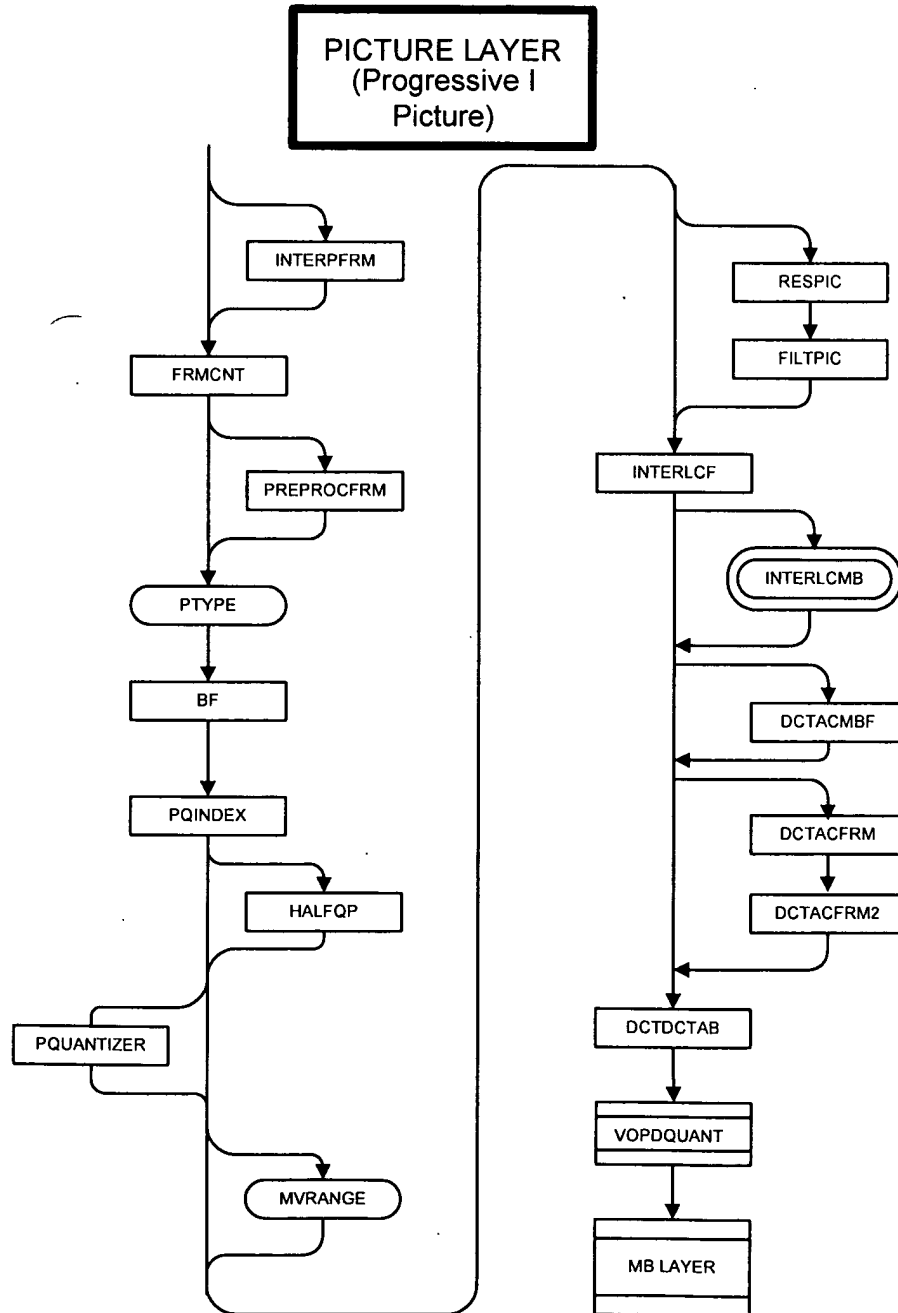
Figure 8: Syntax diagram for the Progressive I picture layer bitstream

**PICTURE LAYER
(Interlace I Picture)**

FRMCNT

PTYPE

BF

PQINDEX

RESPIC2

FILTPIC

INTERLCF

INTERLCMB

DCTACMBF

DCTACFRM

DCTACFRM2

DCTDCTAB

VOPDQUANT

MB LAYER
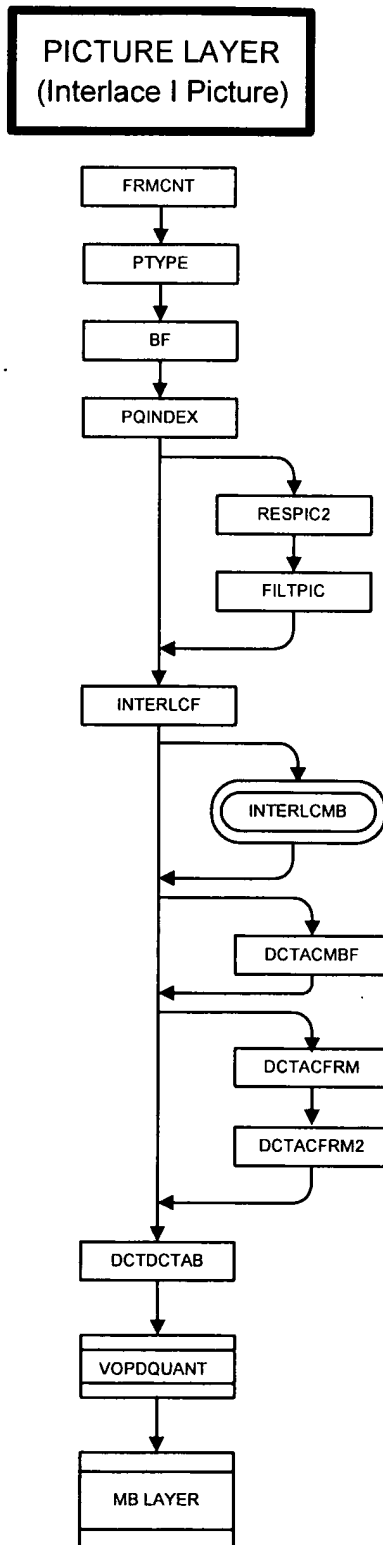
Figure 9: Syntax diagram for the Interlace I picture layer bitstream
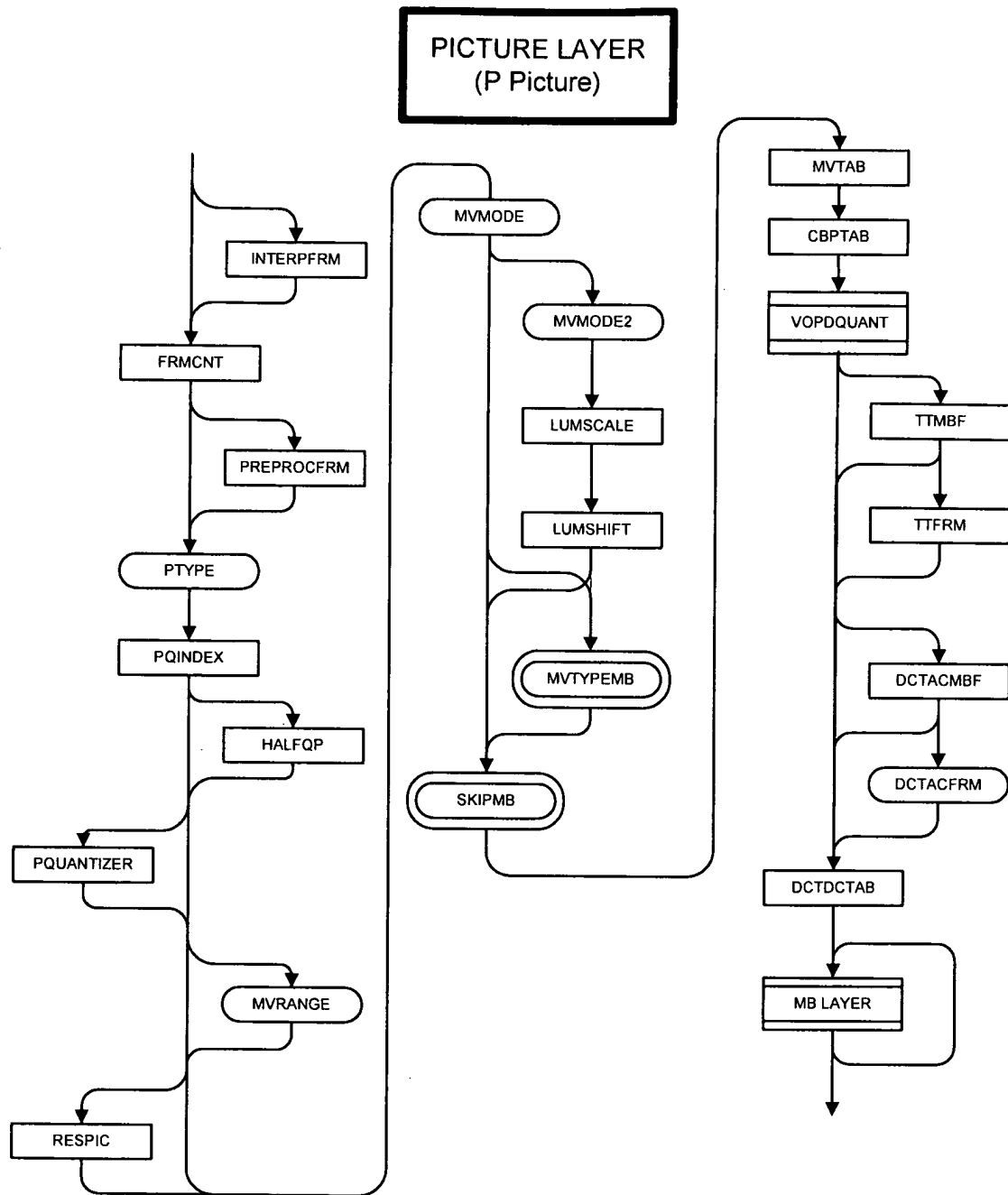
Figure 10: Syntax diagram for the Progressive P picture layer bitstream
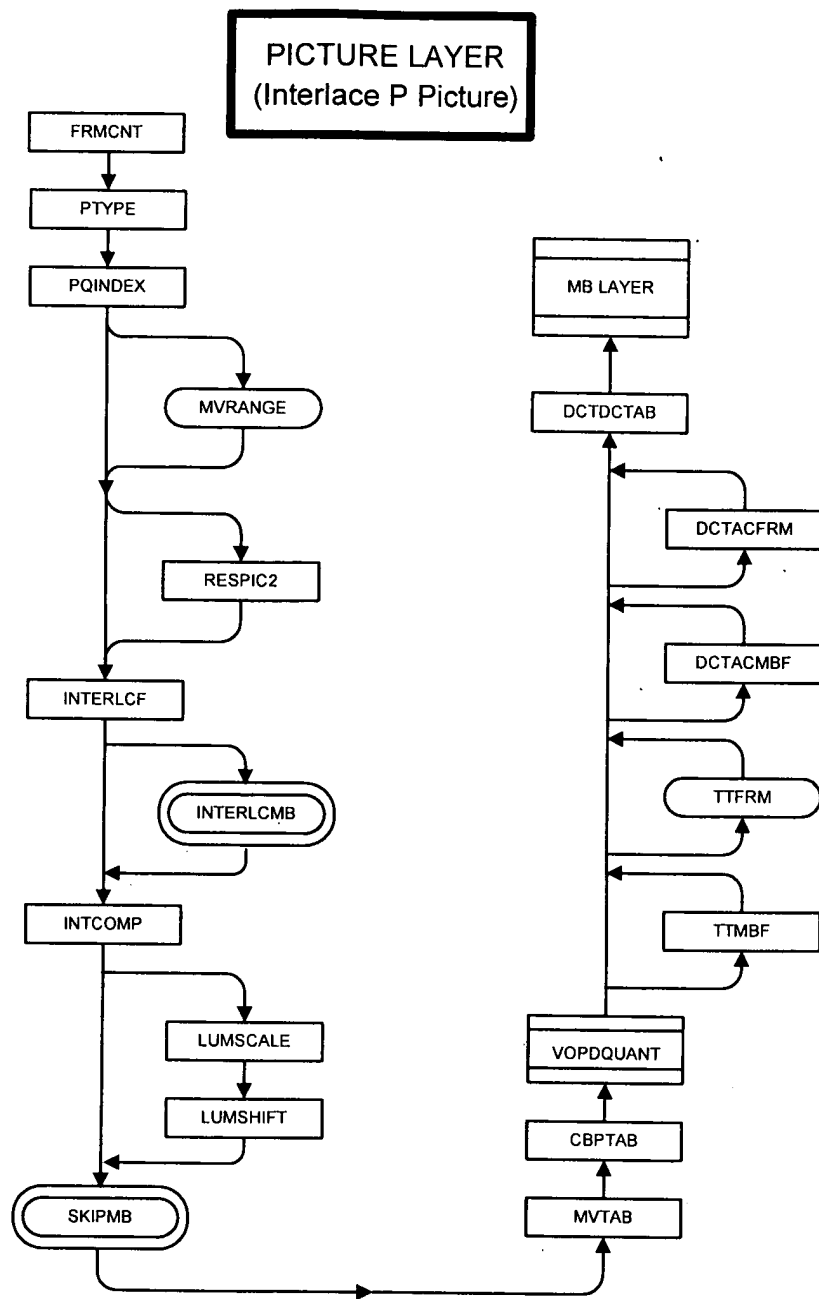
Figure 11: Syntax diagram for the Interlace P picture layer bitstream
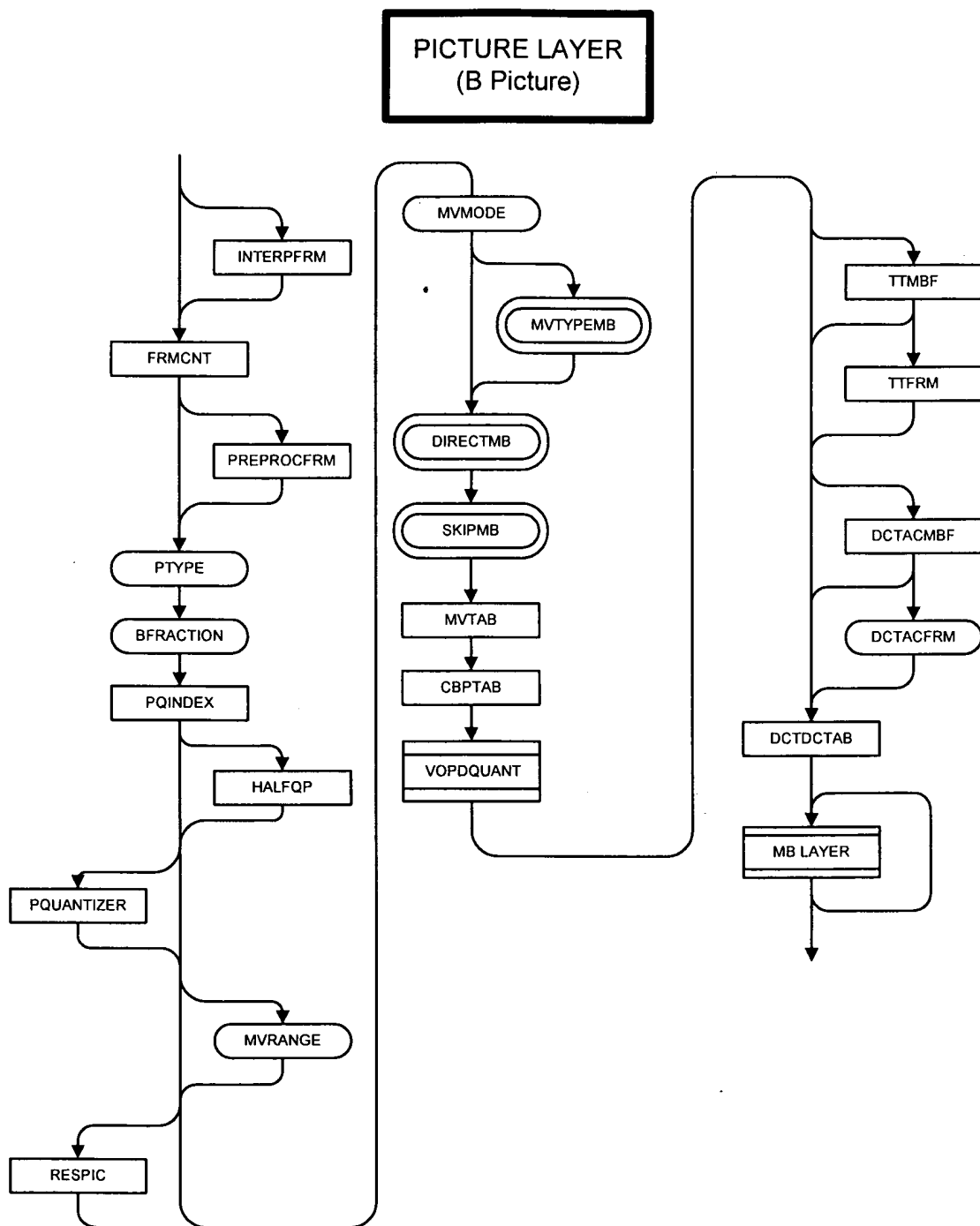
Figure 12: Syntax diagram for the Progressive B picture layer bitstream

PICTURE LAYER
(Interlace B Picture)

INTERPFRM

FRMCNT

PREPROCFRM

PTYPE

BFRACTION

PQINDEX

HALFQP

PQUANTIZER

MVRANGE

RESPIC

INTERLCF

INTERLCMB

DIRECTMB

SKIPMB

MVTAB

CBPTAB

VOPDQUANT

TTMBF

TTFRM

DCTACMBF

DCTACFRM

DCTDCTAB

MB LAYER
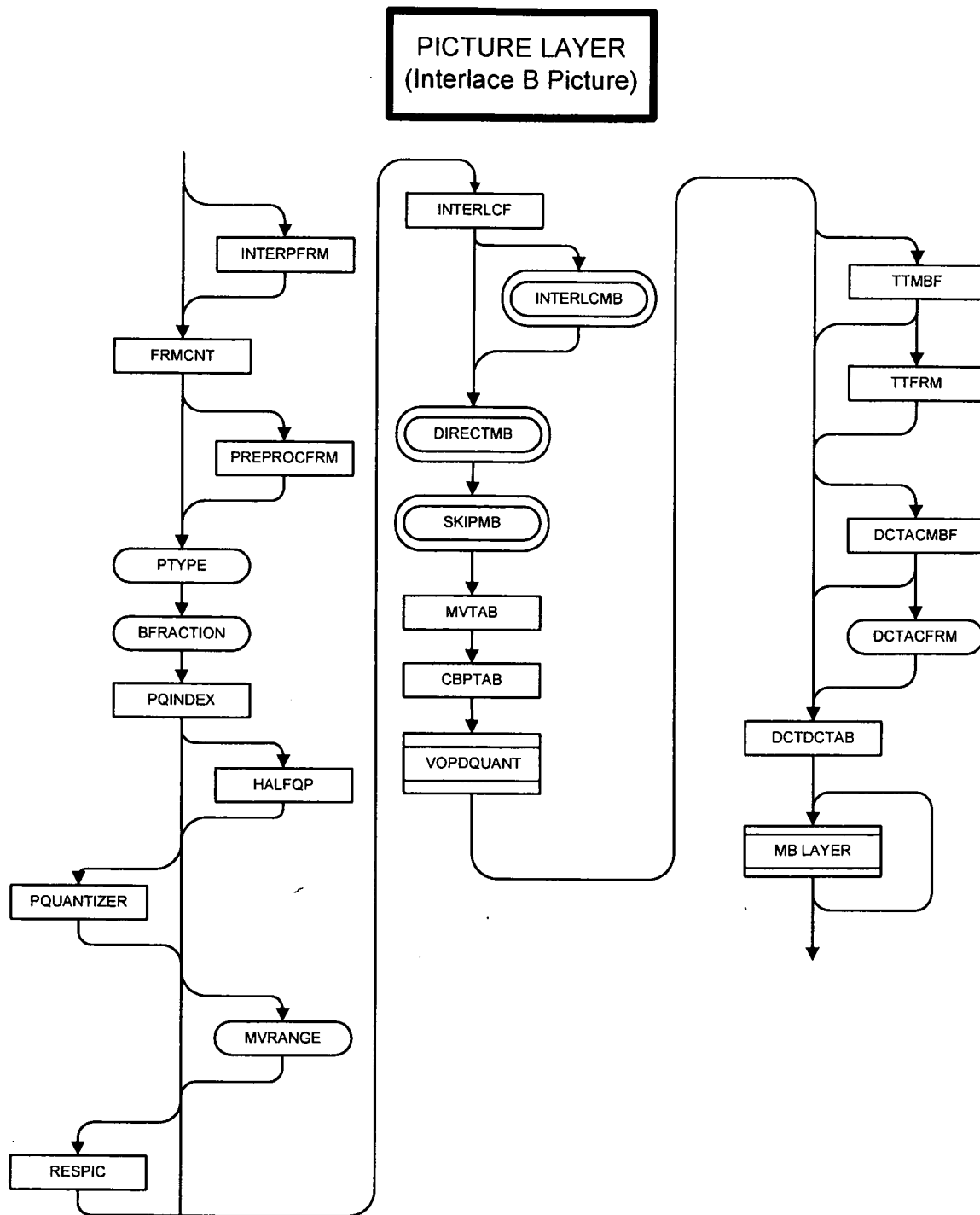
Figure 13: Syntax diagram for the Interlace B picture layer bitstream
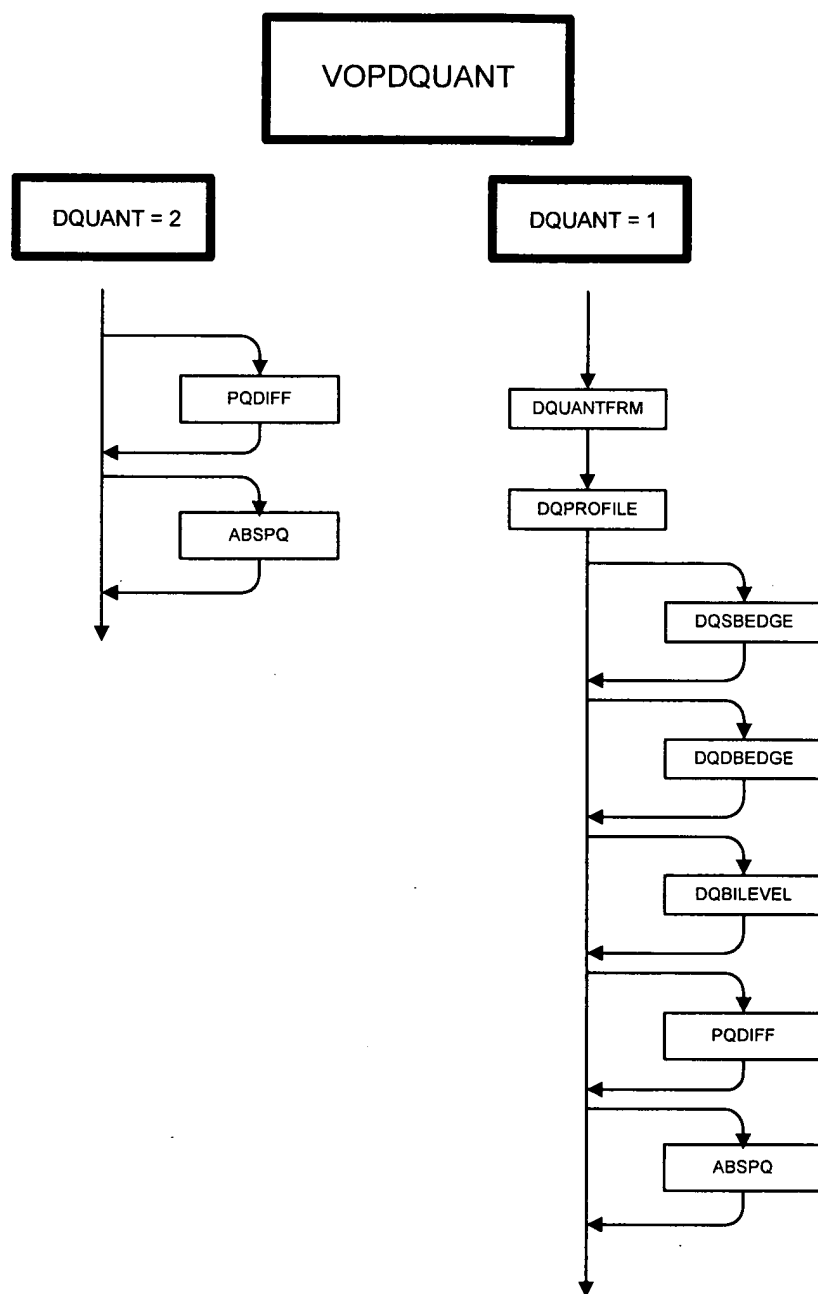
Figure 14: Syntax diagram for VOPDQUANT in (Progressive P, Interlace I and Interlace P) picture header
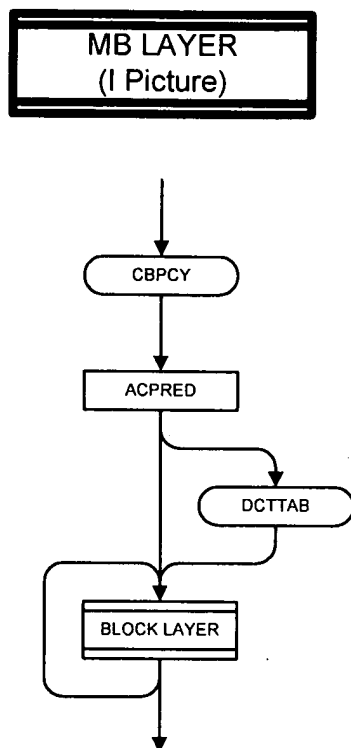
Figure 15: Syntax diagram for macroblock layer bitstream in Progressive-coded I picture

```
┌─────────────────────────┐
║   Intra Frame MB        ║
║      LAYER              ║
└─────────────────────────┘
            │
            ▼
        ( CBPCY )
            │
            ├──────────────┐
            │              ▼
            │      ( SUBBLKPATU )
            │◄─────────────┘
            │              ▼
            │      ( SUBBLKPATV )
            │◄─────────────┘
            ▼
        [ ACPREDMB ]
            │
            ├──────────────┐
            │              ▼
            │        ( MQDIFF )
            │◄─────────────┘
            │              ▼
            │        [ ABSMQ ]
            │◄─────────────┘
            │              ▼
            │        ( DCTTAB )
            │◄─────────────┘
            ▼
    ┌─────────────────┐
    │  INTRA FRAME    │
    │  BLOCK LAYER    │
    └─────────────────┘
```
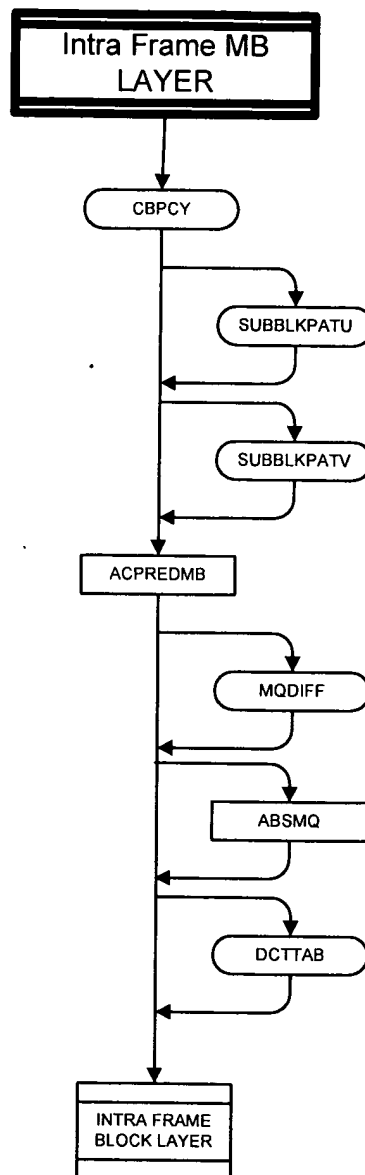
Figure 16: Syntax diagram for intra frame MB layer bitstream in Interlace-coded I picture
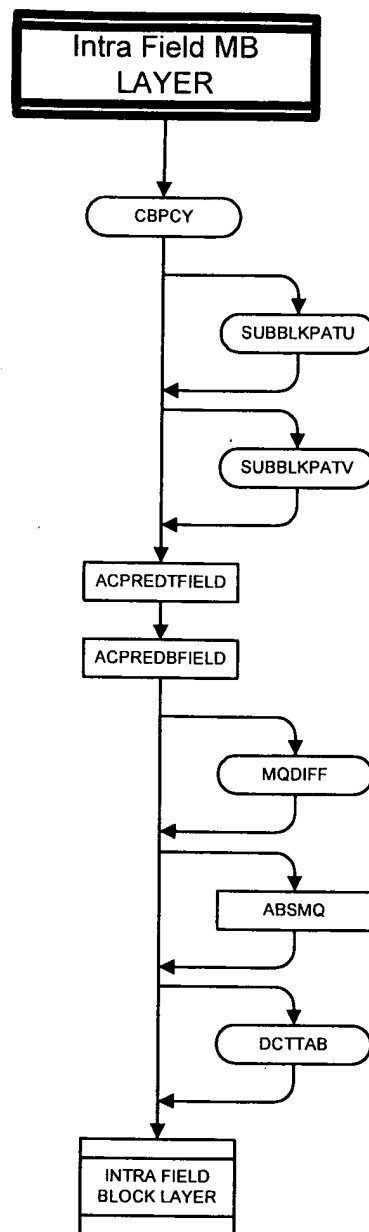
Figure 17: Syntax diagram for intra field MB layer bitstream in Interlace-coded I picture
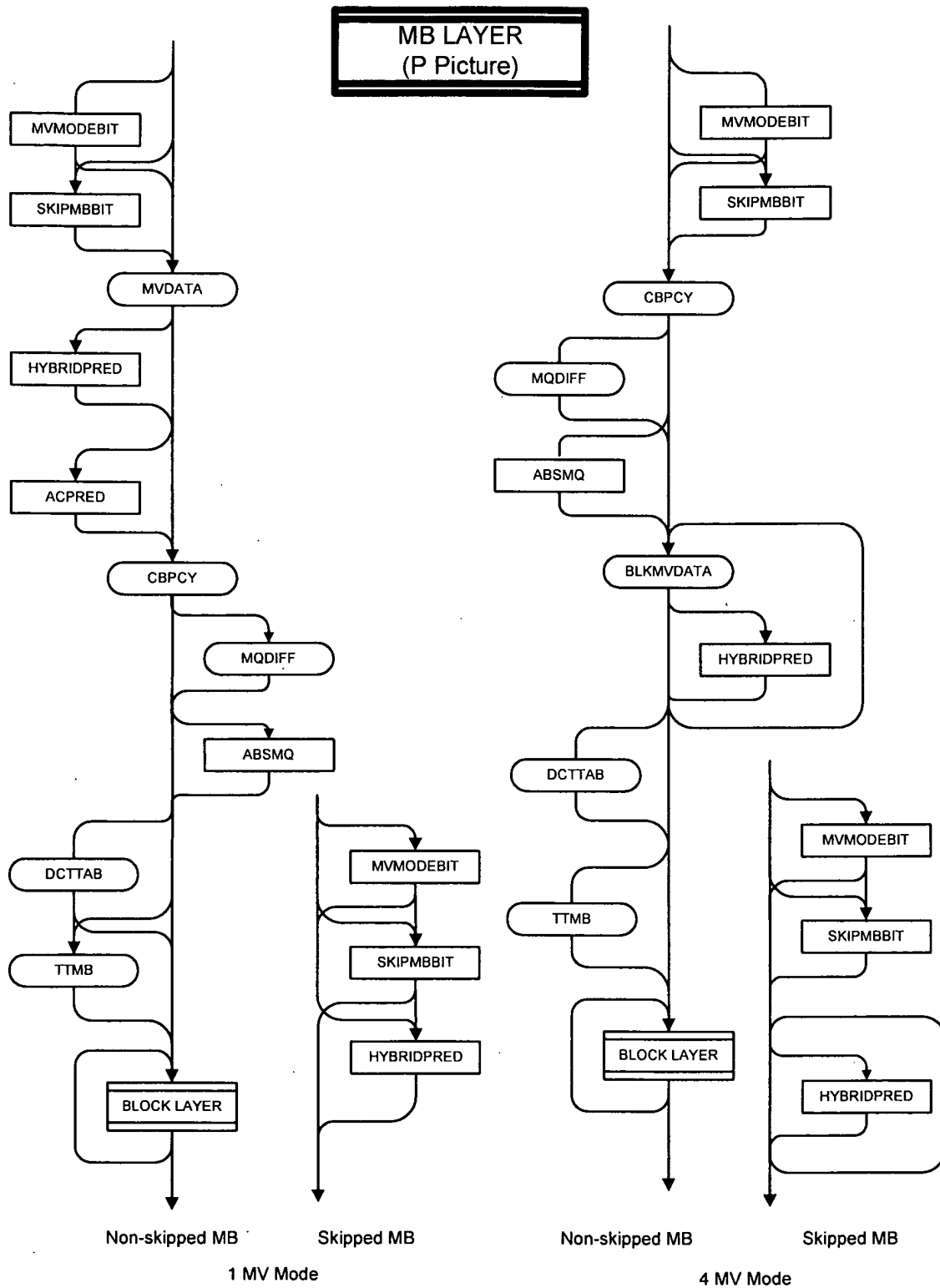
Figure 18: Syntax diagram for macroblock layer bitstream in Progressive-coded P picture
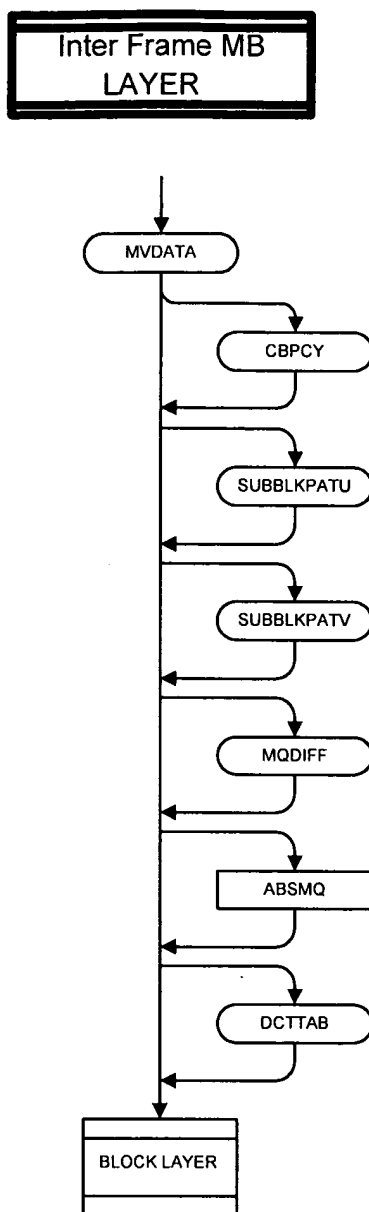
Figure 19: Syntax diagram for frame MB layer in interlace-coded P picture
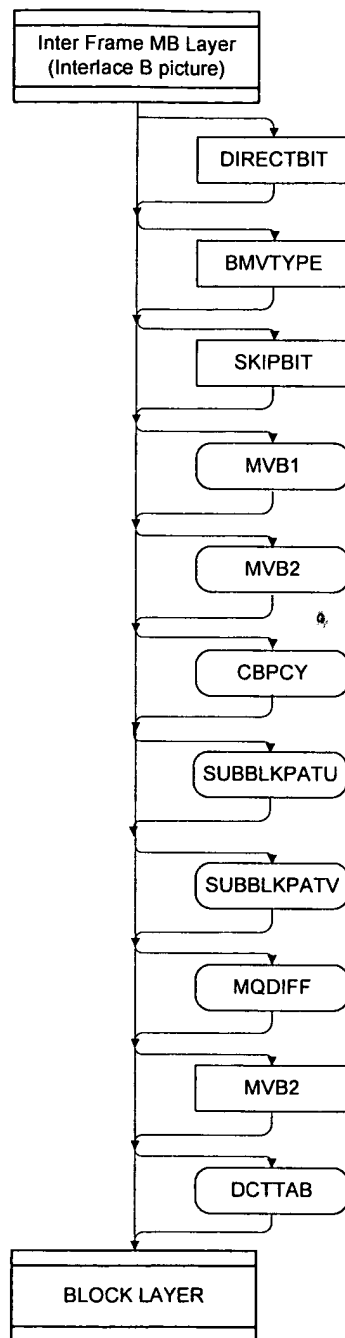
Figure 20: Syntax diagram for frame MB layer in interlace-coded B picture
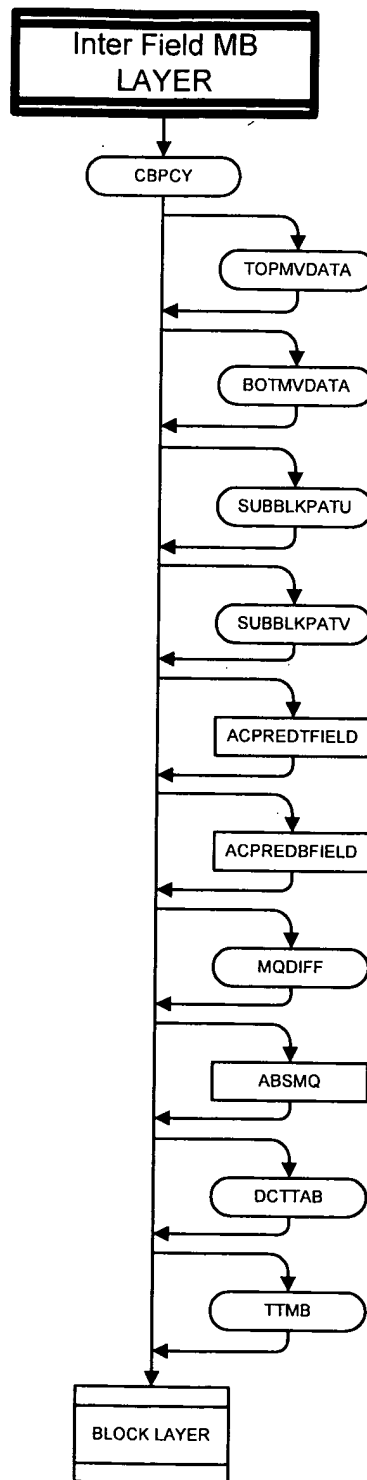
Figure 21: Syntax diagram for field MB layer in interlace-coded P picture
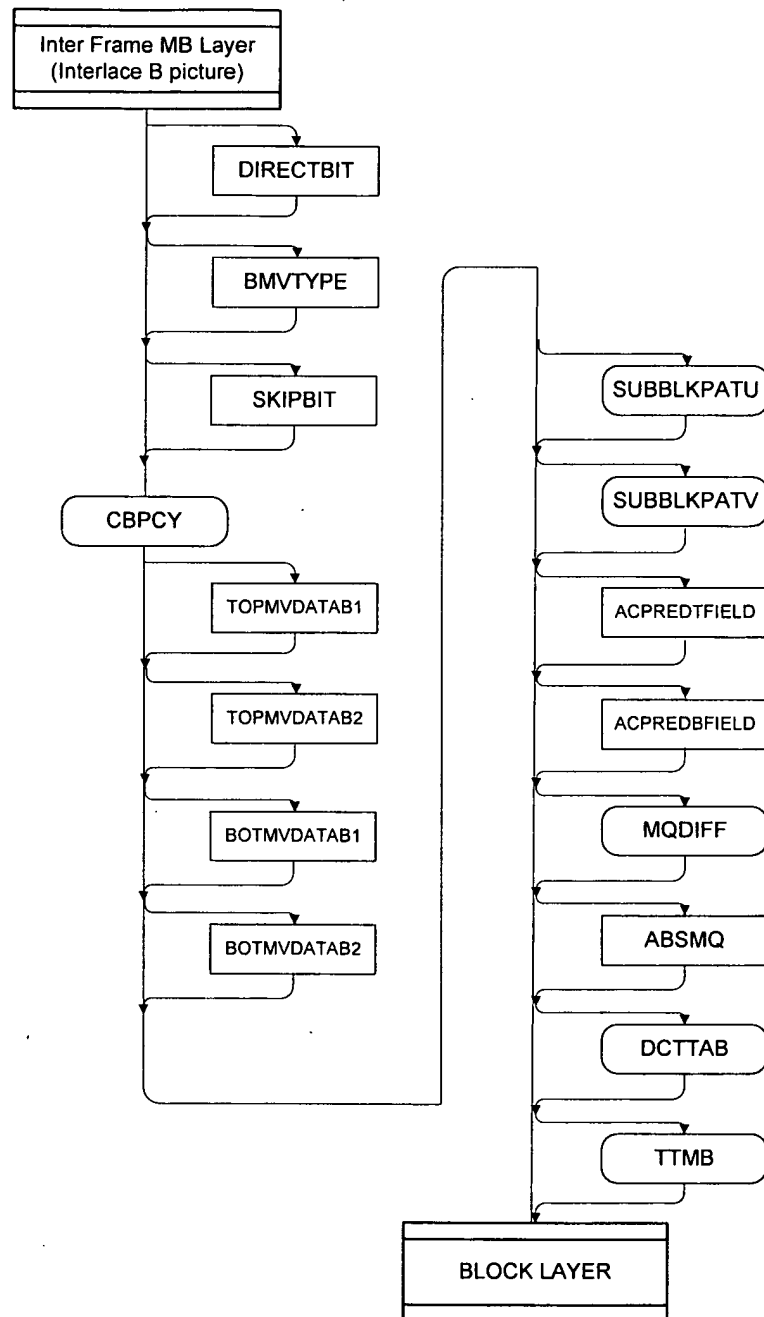
Figure 22: Syntax diagram for field MB layer in interlace-coded B picture
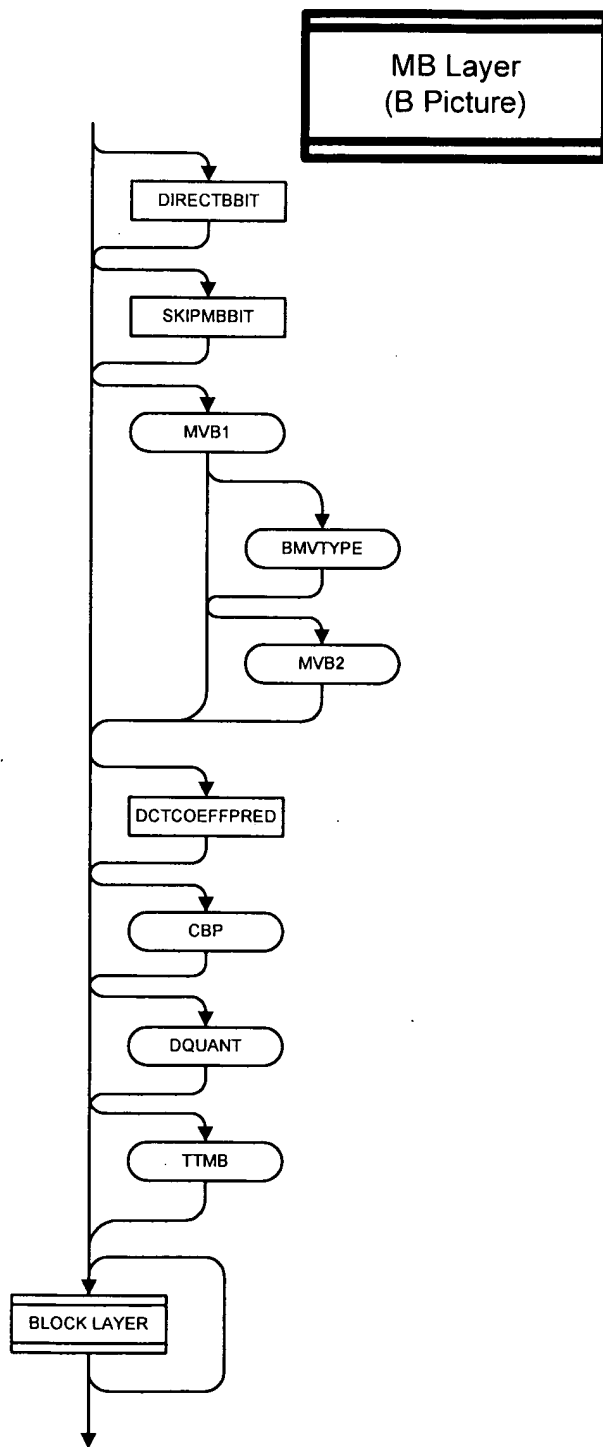
Figure 23: Syntax diagram for macroblock layer bitstream in Progressive-coded B picture
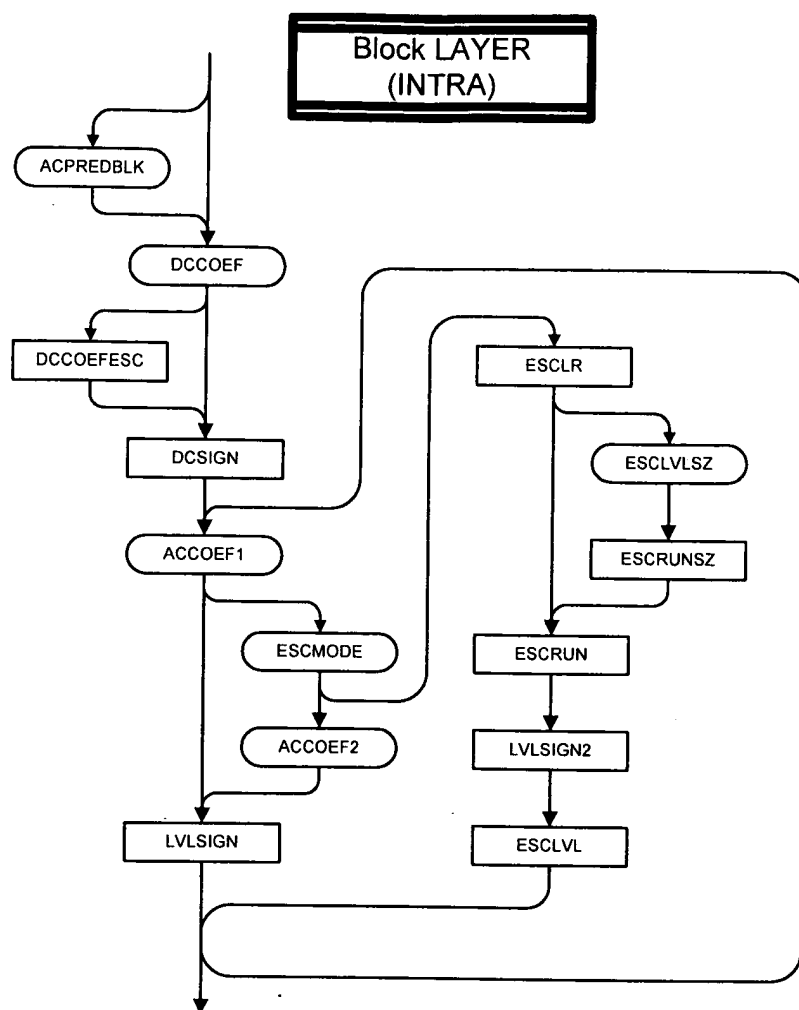
Figure 24: Syntax diagram for the Intra-coded block layer bitstream
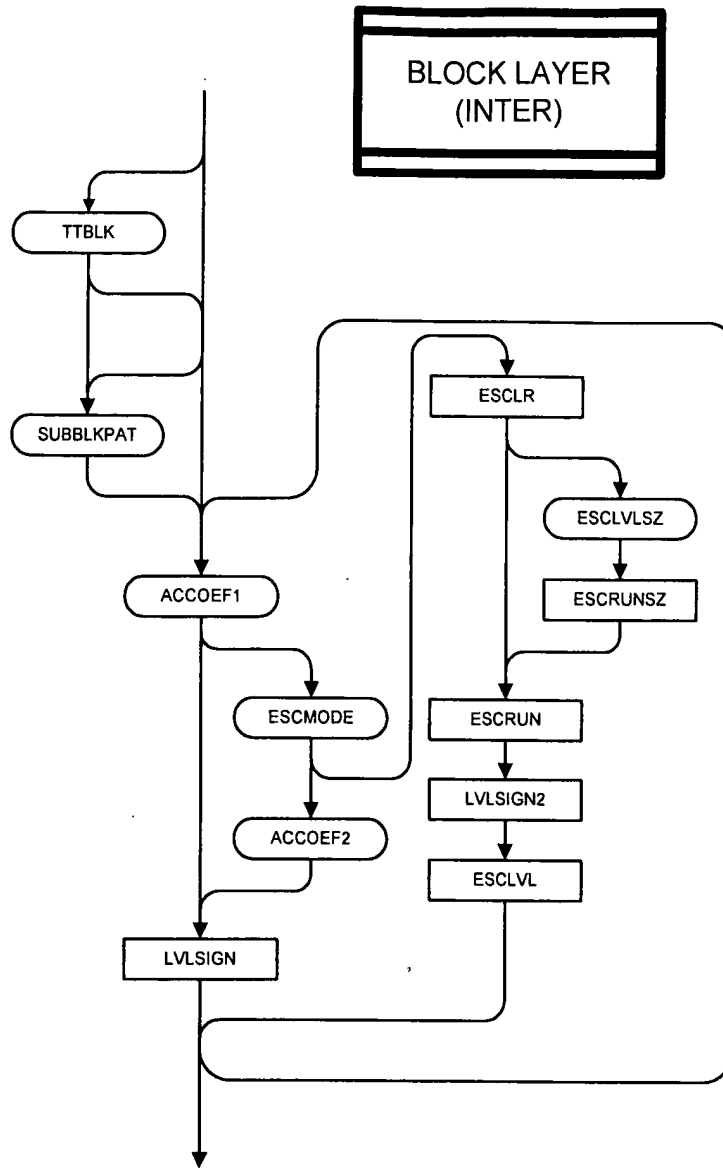
Figure 25: Syntax diagram for the Inter-coded block layer bitstream

3.2.1 Picture layer

Data for each picture consists of a picture header followed by data for the macroblock layer. Figure 8 shows the bitstream elements that make up the I progressive picture layer, Figure 9 shows the bitstream elements that make up the I interlaced picture layer, Figure 10 shows the bitstream elements that make up the P progressive picture layer and Figure 11 shows the bitstream elements that make up the P interlaced picture layer. The following sections give a short description of each of the bitstream elements in the picture layer.

3.2.1.1   Frame Interpolation Flag (INTERPOLATE) (1 bit)

INTERPOLATE is a 1-bit field present in all frame types. The use of this field is currently undefined.

3.2.1.2   Frame Count (FRMCNT) (2 bits)

FRMCNT is a 2-bit field present in I and P picture headers. FRMCNT is used to help with detecting missing frames.

3.2.1.3 Pre-process Frame (PREPROCFRM) (1 bit)

PREPROCFRM is a 1-bit field present in all frame types if the sequence level flag PREPROC = 1 (see section 3.1.17). If PREPROCFRM = 1 then range reduction is used for the frame. If PREPROCFRM = 0 then range reduction is not used for the frame. See sections 4.1.1.7 and 4.4.3.13 for a description of range reduction decoding.

3.2.1.4 Picture Type (PTYPE) (Variable size)

Depending on the value of the sequence level field NUMBFRAMES, PTYPE is either a 1-bit or variable sized field. If NUMBFRAMES = 0 then only I and P frames are present in the sequence and PTYPE is a encoded with according to Table 2.

Table 2: Picture Type FLC if NUMBFRAMES = 0

PTYPE FLC	Picture Type
0	I
1	P

If NUMFRAMES is greater than 0 then B frames are present in the sequence and PTYPE is a variable sized field indicating the picture type of the frame. Table 3 shows the VLC codewords uded to indicate the picture type.

Table 3: Picture Type VLC if NUMFRAMES > 0

PTYPE VLC	Picture Type
1	P
01	I
00	B

3.2.1.5 B Picture Fraction (BFRACTION)(Variable size)

BFRACTION is a VLC symbol that is sent only for B frames. This signals a fraction that can take on a limited set of values between 0 and 1, denoting the relative temporal position of the B frame within the interval formed by its anchors. This fraction is used to scale collocated motion vectors for deriving the direct motion motion vectors.

The mapping of BFRACTION is shown in Table 4. One symbol is unused in the codetable. When BFRACTION is 1111111, this means that the entire B frame is coded independent of its anchors, i.e. as an "Intra" frame. We refer to this frame as an *Intra B frame*. This is not a true I frame since there is no temporal dependency on the Intra B frame, nor does this represent the start of an independently decodable segment.

BFRACTION is sent for both progressive and interlaced sequences.

Table 4: BFRACTION VLC Table

BFRACTION VLC	Fraction	BFRACTION VLC	Fraction
000	1/2	1110101	2/7
001	1/3	1110110	3/7
010	2/3	1110111	4/7
011	1/4	1111000	5/7
100	3/4	1111001	6/7
101	1/5	1111010	1/8
110	2/5	1111011	3/8

1110000	3/5	1111100	5/8
1110001	4/5	1111101	7/8
1110010	1/6	1111110	Invalid
1110011	5/6	1111111	INTRA
1110100	1/7		

3.2.1.6 Buffer Fullness (BF) (7 bits)

BF is a 7-bit field that is only present in I-picture headers. Refer to section 4.1.1.1 for a description of this field.

3.2.1.7 Picture Quantizer Index (PQINDEX) (5 bits)

PQINDEX is a 5-bit field that signals the quantizer scale index for the entire frame. It is present in all picture types. If the implicit quantizer is used (signaled by sequence field QUANTIZER = 00, see section 3.1.19) then PQINDEX specifes both the picture quantizer scale (PQUANT) and the quantizer (3QP ir 5QP deadzone) used for the frame. Table 5 shows how PQINDEX is translated to PQUANT and the quantizer for implicit mode.

Table 5: PQINDEX to PQUANT/Quantizer Deadzone Translation (Implicit Quantizer)

PQINDEX	PQUANT	Quantizer Deadzone	PQINDEX	PQUANT	Quantizer Deadzone
0	NA	NA	16	13	5 QP
1	1	3 QP	17	14	5 QP
2	2	3 QP	18	15	5 QP
3	3	3 QP	19	16	5 QP
4	4	3 QP	20	17	5 QP
5	5	3 QP	21	18	5 QP
6	6	3 QP	22	19	5 QP
7	7	3 QP	23	20	5 QP
8	8	3 QP	24	21	5 QP
9	6	5 QP	25	22	5 QP
10	7	5 QP	26	23	5 QP
11	8	5 QP	27	24	5 QP
12	9	5 QP	28	25	5 QP
13	10	5 QP	29	27	5 QP
14	11	5 QP	30	29	5 QP
15	12	5 QP	31	31	5 QP

If the quantizer is signaled explicitly at the sequence or frame level (signaled by sequence field QUANTIZER = 01, 10 or 11 see section 3.1.19) then PQINDEX is translated to the picture quantizer stepsize PQUANT as indicated by Table 6.

Table 6: PQINDEX to PQUANT Translation (Explicit Quantizer)

PQINDEX	PQUANT 3QP Deadzone	PQUANT 5QP Deadzone	PQINDEX	PQUANT 3QP Deadzone	PQUANT 5QP Deadzzone

0	NA	NA	16	16	14
1	1	1	17	17	15
2	2	1	18	18	16
3	3	1	19	19	17
4	4	2	20	20	18
5	5	3	21	21	19
6	6	4	22	22	20
7	7	5	23	23	21
8	8	6	24	24	22
9	9	7	25	25	23
10	10	8	26	26	24
11	11	9	27	27	25
12	12	10	28	28	26
13	13	11	29	29	27
14	14	12	30	30	29
15	15	13	31	31	31

3.2.1.8 Half QP Step (HALFQP) (1 bit)

HALFQP is a 1bit field present in all frame types if QPINDEX is less than or equal to 8. The HALFQP field allows the picture quantizer to be expressed in half step increments over the low PQUANT range. If HALFQP = 1 then the picture quantizer stepsize is PQUANT + ½. If HALFQP = 0 then the picture quantizer stepize is PQUANT. Therefore, if the 3QP deadzone quantizer is used then half stepsizes are possible up to PQUANT = 9 (ie, PQUANT = 1, 1.5, 2, 2.5 ... 8.5, 9) and then only integer stepsizes are allowable above PQUANT = 9. For the 5QP deadzone quantizer, half stepsizes are possible up to PQUANT = 7 (ie, 1, 1.5, 2, 2.5 ... 6.5, 7).

3.2.1.9 Picture Quantizer Type (PQUANTIZER) (1 bit)

PQUANTIZER is a 1 bit field present in all frame types if the sequence level field QUANTIZER = 01 (see section 3.1.19). In this case, the quantizer used for the frame is specified by PQUANTIZER. If PQUANTIZER = 0 then the 5QP deadzone quantizer is used for the frame. If PQUANTIZER = 1 then the 3QP deadzone quantizer is used.

3.2.1.10 Extended MV Range Flag (MVRANGE) (Variable size)

MVRANGE is a variable-sized field present only in P pictures, and only for sequences coded using the Main Profile with the sequence-layer BROADCAST bit set to 1. The default range of motion vectors is [-64 63.f] X [-32 31.f], where f is the fractional motion vector ¾ for ¼ pixel motion and ½ for ½ pixel motion resolution. In other words, the default range for quarter-pixel motion modes is [-64 63¾] along the horizontal (X-axis) and [-32 31¾] along the vertical (Y-axis). The default range is chosen under Simple Profile, and when BROADCAST is 0 under Main (and Complex) Profile encoding.

Table 7 lists the four possible binary codewords for MVRANGE and the corresponding motion vector range signaled by the codeword. Section 4.4.4.2 details the decoding of differential motion vectors for different ranges specified by MVRANGE.

Table 7: Motion Vector Range Signaled by MVRANGE

Codeword in binary	MV range in full pixel units (horiz x vert)
0 (also default)	[-64, 63.f] x [-32, 31.f]

10	[-64, 63.f] x [-64, 63.f]
110	[-128, 127.f] x [-128, 127.f]
111	[-256, 255.f] x [-256, 255.f]

3.2.1.11 Progressive Picture Resolution Index (RESPIC) (2 bits)

The RESPIC field is always present in progressive I and P pictures if MULTIRES =1 in the sequence layer. This field specifies the scaling factor of the current frame relative to the full resolution frame. Table 8 shows the possible values of the RESPIC field. Refer to section 4.1.1.5 and 4.1.1.6 for a description of the upsampling and downsampling process.

Table 8: Progressive picture resolution code-table

RESPIC FLC	Horizontal Scale	Vertical Scale
00	Full	Full
01	Half	Full
10	Full	Half
11	Half	Half

3.2.1.12 Interlaced Picture Resolution Index (RESPIC2) (1 bit)

The RESPIC2 field is always present in interlaced I and P pictures if MULTIRES =1 in the sequence layer. This field specifies the scaling factor of the current frame relative to the full resolution frame. Table 9 shows the possible values of the RESPIC2 field. Refer to section 4.1.1.5 and 4.1.1.6 for a description of the upsampling and downsampling process.

Table 9: Interlaced picture resolution code-table

RESPIC2 FLC	Horizontal Scale	Vertical Scale
0	Full	Full
1	Half	Full

3.2.1.13 Resampling Filter Type Index (FILTPIC) (1 bit)

The FILTPIC field is present in I pictures if MULTIRES = 1 in the sequence layer. It is not present in P pictures. This field specifies the filter type used to upsample the current frame for display or to upsample or downsample the reference frame if the resolution change occurs at a P picture. Table 10 shows the possible values of the FILTPIC field. Refer to section 4.1.1.6 for a description of the filters used in the upsampling and downsampling process.

Table 10: Resample filter code-table

FILTPIC FLC	Upsample Filter	Downsample Filter
0	10 tap	6 tap
1	3 tap	5 tap

3.2.1.14 X8 Intra Frame (X8IF) (1 bit)

This field is present only in I-picture headers and only if the sequence-level field X8INTRA = 1 (described in section 3.1.7). A value of 0 signals that the I picture uses the baseline decoding method. A value of 1 indicates that the I picture uses the X8INTRA method.

3.2.1.15 Skipped Macroblock Bitfield (SKIPMB)(Variable size)

The SKIPMB field is present only present in P pictures. The SKIPMB field encodes the skipped macroblocks in P pictures using a bitplane coding method. Refer to section 4.10 for a description of the bitplane coding method.

3.2.1.16 B Frame Direct Mode Macroblock Bitfield (DIRECTMB)(Variable size)

The DIRECTMB field is present only present in B pictures. The DIRECTMB field uses bitplane coding to indicate the macroblocks in the B picture that are coded in direct mode. The DIRECTMB field may also signal that the direct mode is signaled in raw mode in which case the direct mode is signaled at the macroblock level (see section 3.2.2.11). Refer to section 4.10 for a description of the bitplane coding method.

3.2.1.17 Motion Vector Mode (MVMODE) (Variable size)

The MVMODE field is present in P and B picture headers. For P Pictures, The MVMODE field signals one of four motion vector coding modes or one intensity compensation mode. Depending on the value of PQUANT, either Table 11 or Table 12 is used to decode the MVMODE field.

Table 11: P Picture Low rate (PQUANT > 12) motion vector mode codetable

MVMODE VLC	Mode
1	1 MV Half-pel bilinear
01	1 MV
001	1 MV Half-pel
0001	Mixed MV
0000	Intensity Compensation

Table 12: P Picture High rate (PQUANT <= 12) motion vector mode codetable

MVMODE VLC	Mode
1	1 MV
01	Mixed MV
001	1 MV Half-pel
0001	1 MV Half-pel bilinear
0000	Intensity Compensation

Intensity compensation is not signaled for B Pictures and only two motion modes are valid. Tables Table 13 and Table 14 show the tables used to to code the motion vector mode for B Pictures. For interlaced sequences, the motion vector mode is always quarter-pel bicubic.

Table 13: B Picture High rate (PQUANT <= 12) motion vector mode codetable

MVMODE VLC	Mode

34

1	Quarter-pel Bicubic
000	Half-pel Bilinear

Table 14: B Picture Low rate (PQUANT > 12) motion vector mode codetable

MVMODE VLC	Mode
1	Half-pel Bilinear
01	Quarter-pel Bicubic

3.2.1.18 Motion Vector Mode 2(MVMODE2) (Variable size)

The MVMODE2 field is only present in P pictures and only if the picture header field MVMODE signals intensity compensation. Refer to section 4.4.3.3 for a description of motion vector mode/intensity compensation. Table 11 and Table 12 are used to decode the MVMODE2 field.

3.2.1.19 Intensity Compensation (INTCOMP) (1 bit)

The INTCOMP field is only present for Interlace P pictures. INTCOMP = 1 signals intensity compensation.

3.2.1.20 Luminance Scale (LUMSCALE)(6 bits)

The LUMSCALE field is only present in P pictures and only if the picture header field MVMODE signals intensity compensation or INTCOMP = 1. Refer to section 4.4.7 for a description of intensity compensation.

3.2.1.21 Luminance Shift (LUMSHIFT)(6 bits)

The LUMSHIFT field is only present in P pictures and only if the picture header field MVMODE signals intensity compensation or INTCOMP = 1. Refer to section 4.4.7 for a description of intensity compensation.

3.2.1.22 Motion Vector Type Bitplane (MVTYPEMB)(Variable size)

The MVTYPE field is present in P pictures if MVMODE or MVMODE2 indicates that Mixed MV motion vector mode is used. The MVTYPEMB field uses bitplane coding to signal the motion vector type (1 or 4 MV) for each macroblock in the frame. Refer to section 4.10 for a description of the bitplane coding method. Refer to section 4.4.4.2 for a description of the motion vector decode process.

3.2.1.23 Interlace Field Coding Status Flag (INTRLCF)(1 bit)

The INTRLCF field is a 1 bit value present in Interlace I and P pictures. INTRLCF signals whether the macroblocks are coded in frame mode or frame/field mode. This field is only present if the sequence level field INTERLACE = 1. If INTRLCF = 0 then all macroblocks are coded in frame mode. If INTRLCF = 1 then the macroblocks may be coded in field or frame mode. In this case (INTRLCF = 1) the INTRLCMB field follows in the bitstream and indicates the frame/field coding status for each macroblock.

3.2.1.24 Field/Frame Mode Bitplane (INTRLCMB)(Variable size)

The INTRLCMB field is present in I and P pictures if INTRLCF indicates that Field/Frame mode is used for the frame (INTRLCF = 1). The INTRLCMB field uses bitplane coding to signal the coding mode (frame or field) for each macroblock in the frame. Refer to section 4.10 for a description of the bitplane coding method.

3.2.1.25 Motion Vector Table (MVTAB) (2 bits)

The MVTAB field is a 2 bit value present only in P frames. The MVTAB field indicates which of four Huffman tables are used to encode the motion vector data. Refer to section 4.4.4.2 for a description of the motion vector decoding process.

Table 15: MVTAB code-table

FLC	Motion Vector Huffman Table

00	Huffman Table 0
10	Huffman Table 1
01	Huffman Table 2
11	Huffman Table 3

The motion vector Huffman tables are listed in section 5.7.

3.2.1.26 Coded Block Pattern Table (CBPTAB) (2 bits)

The CBPTAB field is a 2 bit value present only in P frames. This field signals the Huffman table used to decode the CBPCY field (described in section 4.4.4.2) for each coded macroblock in P-pictures. Refer to section 4.4.3.6 for a description of how the CBP Huffman table is used in the decoding process.

The CBPCY Huffman tables are listed in sections 5.2 and 5.3.

3.2.1.27 Macroblock Quantization (VOPDQUANT) (Variable size)

The VOPDQUANT field is made up of several bitstream syntax elements as shown in Figure 14. VOPDQUANT is present in Progressive P picture and Interlace I and P pictures when the sequence header DQUANT field is nonzero. The syntax of VOPDQUANT is dependent on the picture type (whether it's an I picture or a P picture) and the value of DQUANT.

Case 1: DQUANT = 1.

In this case, we will have four possibilities:

1. The macroblocks located on the boundary are quantized with a second quantization step size (ALTPQUANT) while the rest of the macroblocks are quantized with the frame quantization step size (PQUANT).

2. We will signal two adjacent edges (see Table 18) and those macroblocks located on the two edges are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.

3. We will signal one edge and those macroblock located on the edge are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.

4. Every single macroblock can be quantized differently. In this case, we will indicate whether each macroblock can select from two quantization steps (PQUANT or ALTPQUANT) or each macroblock can be arbitrarily quantized using any step size.

Case 2: DQUANT = 2.

The macroblocks located on the boundary are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.

3.2.1.28 VOPDQUANT Syntax Elements

The new syntax elements are introduced in VOPDQUANT and then we will describe how new syntax varies according to picture type and DQUANT value.

- **DQUANTFRM** (1 bit)

  The DQUANTFRM field is a 1 bit value that is present only when DQUANT = 1. If DQUANTFRM = 0 then the current picture is only quantized with PQUANT.

- **DQPROFILE** (2 bits)

  The DQPROFILE field is a 2 bits value that is present only when DQUANT = 1 and DQUANTFRM = 1. It indicates where we are allowed to change quantization step sizes within the current picture.

Table 16: Macroblock Quantization Profile (DQPROFILE) Code Table

FLC	Location
00	All four Edges

01	Double Edges
10	Single Edges
11	All Macroblocks

- **DQSBEDGE** (2 bits)

The DQSBEDGE field is a 2 bits value that is present when DQPROFILE = Single Edge. It indicates which edge will be quantized with ALTPQUANT.

Table 17: Single Boundary Edge Selection (DQSBEDGE) Code Table

FLC	Boundary Edge
00	Left
01	Top
10	Right
11	Bottom

- **DQDBEDGE** (2 bits)

The DQSBEDGE field is a 2 bits value that is present when DQPROFILE = Double Edge. It indicates which two edges will be quantized with ALTPQUANT.

Table 18: Double Boundary Edges Selection (DQDBEDGE) Code Table

FLC	Boundary Edges
00	Left and Top
01	Top and Right
10	Right and Bottom
11	Bottom and Left

- **DQBILEVEL** (1 bit)

The DQBILEVEL field is a 1 bit valu that is present when DQPROFILE = All Macroblock. If DQBILEVEL = 1, then each macroblock in the picture can take one of two possible values (PQUANT or ALTPQUANT). If DQBILEVEL = 0, then each macroblock in the picture can take on any quantization step size.

- **PQDIFF** (3 bits)

PQDIFF is a 3 bit field that encodes either the PQUANT differential or encodes an escape code.

If PQDIFF does not equal 7 then PQDIFF encodes the differential and the ABSPQ field does not follow in the bitstream. In this case:

ALTPQUANT = PQUANT + PQDIFF + 1

If PQDIFF equals 7 then the ABSPQ field follows in the bitstream and ALTPQUANT is decoded as:

ALTPQUANT = ABSPQ

- **ABSPQ** (5 bits)

ABSPQ is present in the bitstream if PQDIFF equals 7. In this case, ABSPQ directly encodes the value of ALTPQUANT as described above.

3.2.1.29 Macroblock-level Transform Type Flag (TTMBF) (1 bit)

This field is present only in P-picture headers and only if the sequence-level field VSTRANSFORM = 1 (described in section 3.1.9). If TTMBF = 0 then the TTFRM field is also present in the picture layer. See section 4.4.3.8 for a description.

3.2.1.30 Frame-level Transform Type (TTFRM) (2 bits)

This field is present in P-picture and B-frame headers if VSTRANSFORM = 1 and TTMBF = 0. The TTFRM field is decoded using Table 19. See section 4.4.3.9 for a description.

Table 19: Transform type select code-table

FLC	Transform type
00	8x8 DCT
01	8x4 DCT
10	4x8 DCT
11	4x4 DCT

3.2.1.31 Macroblock-level DCT AC Coding Set Flag (DCTACMBF) (1 bit)

The DCTACMBF field is present in I pictures if the sequence-level field DCTTABSWITCH = 1 (described in section 3.1.14) and X8IF = 0. It is present in P pictures if DCTTABSWITCH = 1. This field indicates whether the DCTACFRM and DCTACFRM2 fields are also present in the picture header. See sections 4.1.1.2 and 4.4.3.11 for descriptions.

3.2.1.32 Frame-level DCT AC Coding Set Index (DCTACFRM) (Variable size)

This field is present in I pictures if DCTTABSWITCH = 1, DCTACMBF = 0 and X8IF = 0. It is present in P pictures if DCTTABSWITCH = 1 and DCTACMBF = 0. Table 20 is used to decode the DCTACFRM field. See section 4.4.3.11 for a description of the DCTACFRM field.

Table 20: DCT AC coding set index code-table

VLC	Coding set index
0	0
10	1
11	2

See section 4.1.3.4 for a description of the DCT AC coding sets.

3.2.1.33 Frame-level DCT AC Table-2 Index (DCTACFRM2) (Variable size)

This field is only present in baseline I pictures (X8IF = 0) and only if DCTTABSWITCH = 1 and DCTACMBF = 0. Table 20 is used to decode the DCTACFRM2 field.

See section 4.1.3.4 for a description of the DCT AC coding sets.

3.2.1.34 Intra DCT DC Table (DCTDCTAB) (1 bit)

This field is always present in P pictures and baseline I pictures (X8IF = 0). See section 4.1.1.4 for a description.

3.2.2 Macroblock Layer

Data for each macroblock consists of a macroblock header followed by the block layer. Figure 15 - Figure 21 show the macroblock layer structure for I picture and P picture macroblocks. The elements that make up the macroblock

layer are described in the following sections. The picture types that the macroblock layer syntax elements occur in are indicated in the square brackets.

3.2.2.1 Skip MB Bit (SKIPMBBIT)(1 bit)[P,B]

SKIPMBBIT is a 1-bit field present in P and B frame macroblocks if the frame level field SKIPMB (see section 3.2.1.15) indicates that the raw mode is used. If SKIPMBBIT = 1 then the macroblock is skipped.

3.2.2.2 MV Mode Bit (MVMODEBIT)(1 bit)[P]

SKIPMBBIT is a 1-bit field present in P frame macroblocks if the frame level field MVTYPEMB (see section 3.2.1.22) indicates that the raw mode is used. If MVMODEBIT = 0 then the macroblock is coded in 1MV mode and if MVMODEBIT = 1 then the macroblock is coded in 4MV mode.

3.2.2.3 Coded Block Pattern (CBPCY) (Variable size)[I, P,B]

CBPCY is a variable-length field present in both I picture and P picture macroblock layers. Section 4.1.2.1 describes the CBPCY field in I picture macroblocks and section 4.4.4.2 describes the CBPCY field in P picture macroblocks.

3.2.2.4 AC Prediction Flag (ACPRED)(1 bit)[I, P,B]

The ACPRED field is present in all I picture macroblocks and in 1MV Intra macroblocks in P pictures (see section 4.4.4.1 for a description of the macroblock types). This is a 1-bit field that specifies whether the blocks were coded using AC prediction. ACPRED = 0 indicates that AC prediction is not used. ACPRED = 1 indicates that AC prediction is used. See section 4.1.2.2 for a description of the ACPRED field in I pictures and section 4.4.5.1 for a description of the ACPRED field in P pictures.

3.2.2.5 MB-level DCT AC Coding Set Index (DCTTAB) (Variable size)[I, P,B]

The DCTTAB field is present only if the picture layer field DCTACMBF = 1. Table 20 shows the code-table used to decode the DCTTAB field. See section 4.1.2.3 for a description of the DCTTAB in I picture macroblocks and section 4.4.4.2 for a description of the DCTTAB field in P picture macroblocks.

3.2.2.6 Macroblock Quantizer Differential (MQDIFF)(Variable size)[I,P,B]

MQDIFF is a variable-sized bit field present in Interlace I pictures and P pictures and Progressive P pictures. It is present only if the picture layer field DQPROFILE = All Macroblocks. The syntax depends on the DQBILEVEL field as described below.

If DQBILEVEL = 1, then DIFFMQ is a 1 bit field and the ABSMQ field does not follow in the bitstream. If MQDIFF = 0 then MQUANT = PQUANT (meaning that PQUANT is used as the quantization step size for the current macroblock). If MQDIFF = 1 then MQUANT = ALTPQUANT.

If DQBILEVEL = 0, then MQDIFF is a 3 bit field. In this case MQDIFF decodes either to an MQUANT differential or to an escape code as follows:

If MQDIFF does not equal 7 then MQDIFF encodes the differential and the ABSMQ field does not follow in the bitstream. In this case:

MQUANT = PQUANT + MQDIFF

If MQDIFF equals 7 then the ABSMQ field follows in the bitstream and MQUANT is decoded as:

MQUANT = ABSMQ

3.2.2.7 Absolute Macroblock Quantizer Scale (ABSMQ)(5 bits)[I,P,B]

ABSMQ is present in the bitstream if MQDIFF equals 7. In this case, ABSMQ directly encodes the value of MQUANT as described above.

3.2.2.8 Motion Vector Data (MVDATA)(Variable size)[P]

MVDATA is a variable sized field present in P picture macroblocks. This field encodes the motion vector(s) for the macoblock. See section 4.4.4.2 for a description of the motion vector decode process.

3.2.2.9 Hybrid Motion Vector Prediction (HYBRIDPRED)(1 bit)[P,B]

HYBRIDPRED is a 1-bit field present in P picture macroblocks. Section 4.4.4.2 describes how HYBRIDPRED is used in the decoding process.

3.2.2.10 MB-level Transform Type (TTMB)(Variable size)[P,B]

The TTMB field is a variable length field present in P and B picture macroblocks if the picture layer field TTMBF = 1. As shown in tables Table 21, Table 22 and Table 23, the TTMB field specifies the transform type, the signal level and the subblock pattern. If the signal type speficies macroblock mode the transform type decoded from the TTMB field is used to decode all coded blocks in the macroblock. If the signal type signals block mode then the transform type decoded from the TTMB field is used to decode the first coded block in the macroblock. The transform type of the remaining blocks is coded at the block level. If the transform type is 8x4 or 4x8 then the subblock pattern indicates the subblock pattern of the first block.

The table used to decode the TTMB field depends on the value of PQUANT. For PQUANT less than or equal to 4, Table 21 is used. For PQUANT greater than 4 and less than or equal to 12, Table 22 is used. For PQUANT greater than 12, Table 23 is used.

The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

Table 21: High Rate (PQUANT < 5) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
00	8x4	Block	Both
01	8x8	Block	NA
11	4x8	Block	Both
100	4x4	Block	NA
10100	8x8	Macroblock	NA
10101	4x8	Block	Left
10110	4x8	Block	Right
101110	8x4	Block	Bottom
1011111	8x4	Block	Top
101111001	8x4	Macroblock	Top
101111010	4x4	Macroblock	NA
101111011	8x4	Macroblock	Both
1011110001	8x4	Macroblock	Bottom
10111100001	4x8	Macroblock	Both
101111000000	4x8	Macroblock	Right
101111000001	4x8	Macroblock	Left

Table 22: Medium Rate (5 =< PQUANT < 13) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
10	8x8	Macroblock	NA
000	4x8	Block	Both
010	4x4	Block	NA
110	8x8	Block	NA
0011	8x4	Block	Top

0110	8x4	Block	Bottom
0111	8x4	Block	Both
1110	4x8	Block	Left
1111	4x8	Block	Right
001001	4x8	Macroblock	Left
001011	8x4	Macroblock	Both
0010001	8x4	Macroblock	Top
0010100	8x4	Macroblock	Bottom
0010101	4x8	Macroblock	Both
00100000	4x8	Macroblock	Left
00100001	4x4	Macroblock	NA

Table 23: Low Rate (PQUANT >= 13) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
10	8x8	Macroblock	NA
000	8x4	Block	Bottom
010	4x8	Block	Right
011	4x8	Block	Left
110	8x8	Block	NA
0011	4x8	Block	Both
1110	8x4	Block	Top
1111	4x4	Block	NA
00101	8x4	Block	Both
001001	8x4	Macroblock	Both
0010001	4x8	Macroblock	Both
00100001	8x4	Macroblock	Top
001000001	4x8	Macroblock	Left
0010000001	8x4	Macroblock	Bottom
00100000000	4x4	Macroblock	NA
00100000001	4x8	Macroblock	Right

3.2.2.11 Direct B Frame Coding Mode (DIRECTBBIT)(1 bit)[B]

DIRECTBBIT is a 1-bit field present in B frame macroblocks if the frame level field DIRECTMB (see section 3.2.1.16) indicates that the raw mode is used. If DIRECTBBIT = 1 then the macroblock is coded using direct mode.

3.2.2.12 B Macroblock Motion Vector 1 (BMV1)(Variable size)[B]

BMV1 is a variable sized field present in B picture macroblocks. This field encodes the first motion vector for the macoblock. See section 4.4.4.2 for a description of the motion vector decode process.

3.2.2.13 B Macroblock Motion Vector 2 (BMV2)(Variable size)[B]

BMV2 is a variable sized field present in B picture macroblocks if the Interpolation mode is used. This field encodes the second motion vector for the macoblock. See section 4.4.4.2 for a description of the motion vector decode process.

3.2.2.14 B Macroblock Motion Prediction Type (BMVTYPE)(Variable size)[B]

BMVTYPE is a variable sized field present in B frame macroblocks that indicates whether the macroblock uses forward, backward or interpolated prediction. As Table 24 shows, the value of BFRACTION (in the picture header, see section 3.2.1.5) along with BMVTYPE determine whether forward or backward prediction are indicated.

Table 24: B Frame Motion Prediction Type

BMVTYPE VLC	Motion Prediction Type	
	BFRACTION <= 1/2	BFRACTION > 1/2
0	Backward	Forward
10	Forward	Backward
11	Interpolated	Interpolated

3.2.3 Block Layer

Figure 24 and Figure 25 show the block layer elements for intra and inter-coded blocks respectively. The elements that make up the block layer are described in the following sections. Specified in square brackets are the types (intra, inter or both) in which the block elements occur.

3.2.3.1 Block AC Prediction Flag (ACPREDBLK)(Variable size)

The ACPREDBLK field is only present in P picture intra-coded blocks and only under certain conditions. See section 4.4.5.1 for a description of when and how the ACPREDBLK field is used.

3.2.3.2 DCT DC Coefficient (DCCOEF)(Variable size)[intra]

The DCCOEF field is only present in intra-coded blocks. This is a variable-length codeword that encodes the DCT DC differential. Refer to section 4.1.3.1 for a description of the DCT DC decoding process. One of two code tables is used to encode the DC differentials (the table is signaled in the DCTDCTAB field in the picture header as described in section 4.1.1.4). Section 5.4 lists the DC Huffman tables.

3.2.3.3 DCT DC Coefficient (DCCOEFESC)(variable size)[intra]

The DCCOEFESC field is only present in intra-coded blocks and only if DCCOEF decodes to the escape code. The size of DCCOEFESC field can be 8, 9 or 10 bits depending on the quantization step size of the block. Refer to section 4.1.3.1 for a description of the DCT DC decoding process.

3.2.3.4 DCT DC Sign (DCSIGN)(1 bit)[intra]

DCSIGN is a one-bit value that indicates the sign of the DC differential. If DCSIGN = 0 then the DC differential is positive. If DCSIGN = 1 then the DC differential is negative.

3.2.3.5 DCT AC Coefficient 1 (ACCOEF1)(Variable size)[both]

ACCOEF1 is present in both intra and inter blocks. This is a variable-length codeword that encodes the run, level and last flag for each non-zero AC coefficient. Refer to section 4.1.3.4 for a description of the DCT AC decoding process. One of three code tables is used to encode ACCOEF1. The table is signaled in the picture or macroblock headers as described in sections 3.2.1.31-3.2.1.33 and 3.2.2.5. Section 5.5 lists the AC Huffman tables.

. 3.2.3.6 DCT AC Coefficient 2 (ACCOEF2)(Variable size)[both]

ACCOEF2 can be present in both intra and inter blocks. It is only present if ACCOEF1 decodes to the escape code and if the ESCMODE field (described in section 3.2.3.7) specifies AC decoding escape mode 1 or 2 (refer to section 4.1.3.4 for a description of the DCT AC decoding process). One of three code tables is used to encode ACCOEF2. The table is signaled in the picture or macroblock headers as described in sections 3.2.1.23-3.2.1.33 and 3.2.2.5. Section 5.5 lists the AC Huffman tables.

3.2.3.7 DCT AC Escape Decoding Mode (ESCMODE)(Variable size)[both]

ESCMODE can be present in both intra and inter blocks. It is only present if ACCOEF1 decodes to the escape code. ESCMODE is a variable-length codeword that signals which of three escape decoding methods are used to decode the AC coefficient. Table 25 shows the code-table used to encode the escape modes.

Table 25: AC escape decoding mode code-table

ESCMODE VLC	AC Escape Decoding Mode
1	Mode 1
01	Mode 2
00	Mode 3

If mode 1 or mode 2 decoding is specified then the bitstream contains the ACCOEF2 element as described in section 3.2.3.6. If mode 3 is specified then the bitstream contains the ESCLR, ESCRUN, ESCLVL and LVLSIGN2 elements and may contain the ESCLVLSZ and ESCRUNSZ elements as described in sections 3.2.3.13 and 3.2.3.14.

3.2.3.8 DCT AC Level Sign (LVLSIGN)(1 bit)[both] .

LVLSIGN can be present in both intra and inter blocks. It will always be present unless ESMODE specifies AC decoding mode 3. LVLSIGN is a one-bit value that specifies the sign of the AC level. Refer to section 4.1.3.4 for a description of the DCT AC decoding process. If LVLSIGN = 0 then the level is positive. If LVSIGN = 1 then the level is negative.

3.2.3.9 Escape Mode 3 Last Run (ESCLR)(1 bit)[both]

ESCLR can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3. ESCLR is a one-bit value that specifies whether this coefficient is the last non-zero coefficient in the block. If ESCLR = 1 then this is the last non-zero coefficient. If ESCLR = 0 then this is not the last non-zero coefficient.

3.2.3.10 Escape Mode 3 Run (ESCRUN)(Calculated size)[both]

ESCRUN can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3. The size of the ESCRUN codeword is fixed throughout the frame with the size being specified in the ESCRUNSZ field described in section 3.2.3.14. ESCRUN directly encodes the run value for the coefficient. For example if the size (from ESCRUNSZ) is 4 bits and the value is [0101] then the run is decoded as 5.

3.2.3.11 Escape Mode 3 Level (ESCLVL)(Calculated size)[both]

ESCLVL can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3. The size of the ESCLVL codeword is fixed throughout the frame with the size being specified in the ESCLVLSZ field described in section 3.2.3.13. ESCLVL directly encodes the level value for the coefficient. For example if the size (from ESCLVLSZ) is 3 bits and the value is [110] then the run is decoded as 6.

3.2.3.12 Escape Mode 3 Level Sign (LVLSGN2)(1 bit)[both]

LVLSGN2 can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3. LVLSGN2 is a one-bit value that specifies the sign of the decoded level value (ESCLVL). If LVLSGN2 = 0 then level is positive. If LVLSGN2 = 1 then level is negative.

3.2.3.13 Escape Mode 3 Level Size (ESCLVLSZ)(Variable size)[both]

ESCLVLSZ can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3 and if this is the first time mode 3 has been signaled within the current frame (in other words, all subsequent instances of escape mode 3 coding within this frame do not have this field). ESCLVLSZ is used to specify the codeword size for the mode 3 escape-coded level values for the entire frame. Two different VLC tables are used to encode ESCLVLSZ depending on the value of PQUANT. The two tables are as follows:

Table 26: Escape mode 3 level codeword size code-table for 1 <= PQUANT <= 7

1 <= PQUANT <= 7	
ESCLVLSZ VLC	Level codeword size
001	1
010	2
011	3
100	4
101	5
110	6
111	7
0000	8
0001	9

Table 27: Escape mode 3 level codeword size code-table for 8 <= PQUANT <= 31

8 <= PQUANT <= 31	
ESCLVLSZ VLC	Level codeword size
1	2
01	3
001	4
0001	5
00001	6
000001	7

000000	8

3.2.3.14 Escape Mode 3 Run Size (ESCRUNSZ)(2 bits)[both]

ESCRUNSZ can be present in both intra and inter blocks. It is only present if ESMODE specifies AC decoding escape mode 3 and is only present the first time escape mode 3 is signaled within the frame. ESCRUNSZ is used to specify the codeword size for the mode 3 escape-coded run values for the entire frame. The run codeword size is encoded according to Table 28:

Table 28: Escape mode 3 run codeword size code-table

ESCRUNSZ FLC	Run codeword size
00	3
01	4
10	5
11	6

3.2.3.15 Block-level Transform Type (TTBLK)(Variable size)[inter]

The TTBLK field is present only in inter-coded blocks and only if the macroblock level field TTMB (see section 3.2.2.10) indicates that the signaling level is Block. The 8x8 error blocks can be transformed using an 8x8 DCT, two 8x4 DCTs, two 4x8 DCTs or four 4x4 DCTs. The TTBLK field codes the transform type for the block as well as the subblock pattern if the transform type is 8x4 or 4x8. The table used to decode the TTBLK field depends on the value of PQUANT. If PQUANT is less than or equal to 4 then Table 29 is used. If PQUANT is greater than 4 and less than or equal to 12 then Table 30 is used. If PQUANT is greater than 12 then Table 31 is used. The TTBLK field is not present for the first block in each macroblock since the transform type and subblock pattern decoded in TTMB is used for the first block. TTBLK is present for each coded block after the first. The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

Table 29: High Rate (PQUANT < 5) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
00	8x4	Both
01	4x8	Both
11	8x8	NA
101	4x4	NA
10000	8x4	Top
10001	8x4	Bottom
10010	4x8	Right
10011	4x8	Left

Table 30: Medium Rate (5 =< PQUANT < 13) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
11	8x8	NA
000	4x8	Right

001	4x8	Left
010	4x4	NA
011	8x4	Both
101	4x8	Both
1000	8x4	Bottom
1001	8x4	Top

Table 31: Low Rate (PQUANT >= 13) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
01	8x8	NA
000	4x8	Both
001	4x4	NA
100	8x4	Bottom
110	4x8	Right
111	4x8	Left
1010	8x4	Both
1011	8x4	Top

3.2.3.16 Transform sub-block pattern (SUBBLKPAT)(Variable size)[inter]

The SUBBLKPAT field is only present in inter-coded blocks and only if the transform type for the block is 8x4, 4x8 or 4x4.

For 4x4 transform types, the SUBBLKPAT field indicates which of the 4 4x4 subblocks have at least one non-zero coefficient.
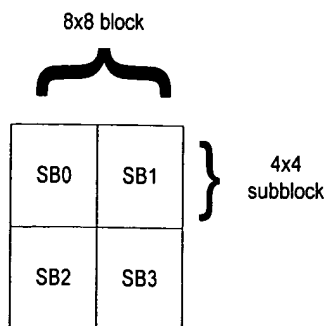
Figure 26: 4x4 Subblocks

The subblock pattern is coded as a 4 bit field where each bit indicates whether the corresponding subblock contains at least one non-zero coefficient. Figure 26 shows the labeling of the 4 subblocks that make up an 8x8 block. The subblock pattern is coded is coded as follows:

Subblock pattern = 8 * SB0 + 4 * SB1 + 2 * SB2 + SB3

46

Where:

SBx = 0 if the corresponding subblock does not contain any non-zero coefficients, and

SBx = 1 if the corresponding subblock contains at least one non-zero coefficient.

The following tables show the VLC codewords used to encode the subblock pattern. The table used depends on the value of PQUANT. If PQUANT is less then or equal to 4 then Table 32 is used. If PQUANT is greater than 4 and less than or equal to 12 then Table 33 is used. If PQUANT is greater than 12 then Table 34 is used.

Table 32: High Rate (PQUANT < 5) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
1	15	01010	8
0000	11	01011	4
0001	13	01100	2
0010	7	01110	1
00110	12	01111	14
00111	3	011010	6
01000	10	011011	9
01001	5		

Table 33: Medium Rate (5 =< PQUANT < 13) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
01	15	1111	4
000	2	00100	6
0011	12	00101	9
1000	3	10110	14
1001	10	10111	7
1010	5	11000	13
1101	8	11001	11
1110	1		

Table 34: Low Rate (PQUANT >= 13) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
010	4	1111	15
011	8	00000	6
101	1	00001	9
110	2	10010	14
0001	12	10011	13
0010	3	11100	7

0011	10	11101	11
1000	5		

8x8 block   8x4 DCT   4x8 DCT
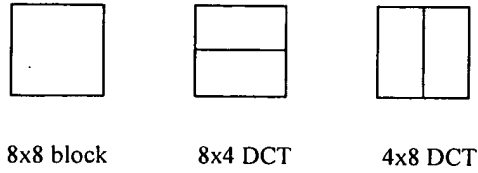
Figure 27: 8x4 and 4x8 Subblocks

For 8x4 or 4x8 transform types, the SUBBLKPAT field specifies which of the two sub-blocks have at least one non-zero coefficient. The data is encoded with the following VLC tables (one for progressive and one for interlace pictures) (an X indicates that the sub-block contains at least one non-zero coefficient):

Table 35: 8x4 and 4x8 Transform sub-block pattern code-table for Progressive pictures

SUBBLKPAT VLC	8x4 Sub-block pattern		4x8 Sub-block pattern	
	Top	Bottom	Left	Right
0		X		X
10	X	X	X	X
11	X		X	

Table 36: 8x4 and 4x8 Transform sub-block pattern code-table for Interlace pictures

SUBBLKPAT VLC	8x4 Sub-block pattern		4x8 Sub-block pattern	
	Top	Bottom	Left	Right
0	X	X	X	X
10		X		X
11	X		X	

3.2.3.17 Block-level Motion Vector Data (BLKMVDATA)(Variable size)|inter|

BLKMVDATA is a field that contains motion information for the block. It is a variable sized field and is only present in certain situations. See section 4.4.4.1 for a description of when the BLKMVDATA field is present and how it is used.

3.3 Bitplane Coding Syntax

Various frame-level syntax elements use a bitplane coding scheme to indicate the status of the macroblocks that make up the frame. For example, in P and B frames, the presence of skipped macroblocks is signaled with a bit set to 0 and the presence of a non-skipped macroblock is signaled with a bit set to 1. These bits are coded as a frame-level bitplane. The following diagram shows the elements that make up the bitplane.
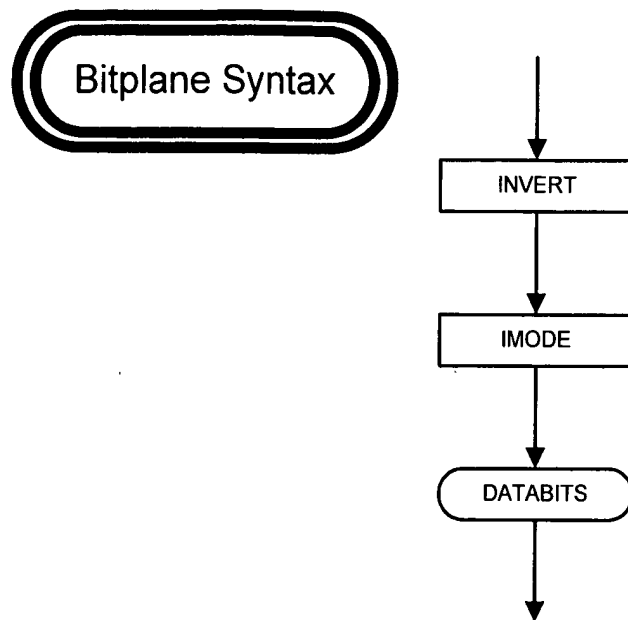
Figure 28: Syntax diagram for the bitplane coding

3.3.1 Invert Flag (INVERT)

The INVERT field is a 1-bit value. Refer to section 4.10.1 for a description of how the INVERT value is used in decoding the bitplane.

3.3.2 Coding Mode (IMODE)

The IMODE field is a variable length value that indicates the coding mode used to encode the bitplane. Table 37 shows the codetable used to encode the IMODE field.

Table 37: IMODE VLC Codetable

IMODE VLC	Coding Mode
10	Norm-2
11	Norm-6
010	Rowskip
011	Colskip
001	Diff-2
0001	Diff-6
0000	Raw

3.3.3 Bitplane Coding Bits (DATABITS)

The DATABITS field is variable sized field that encodes the bitplane. The method used to encode the bitplane is determined by the value of IMODE. Refer to section 4.10.3 for a description the different coding methods.

4 Decoding Process

This section describes the decoding process for baseline I pictures, X8INTRA I pictures, Interlace I pictures, Progressive P pictures, and Interlace P pictures

For I pictures, X8INTRA I picture is used when signaled by X8INTRA = 1 and X8IF = 1. Interlace I picture is used when signaled by INTERLACE = 1. Otherwise, Baseline I picture is used.

For P pictures, Interlace P picture is used when signaled by INTERLACE = 1. Otherwise Progressive P picture is used.

4.1 Baseline I Frame Decoding

The following sections describe the process for decoding baseline I pictures.

4.1.1 Baseline I Picture Layer Decode

Figure 8 shows the elements that make up the baseline I picture layer header. Some of the elements are self-explanatory. The following sections provide extra detail for some of the elements.

4.1.1.1 Buffer Fullness

The BF field is currently undefined.

4.1.1.2 Macroblock-level DCT AC Table Flag

DCTACMBF is a one-bit field that signals whether the coding sets that are used to decode the DCT AC coefficients are signaled at the frame level or macroblock level. If DCTACMBF = 1 then the coding sets used to decode all the AC coefficients in that macroblock are signaled at the macroblock level. If DCTACMBF = 0 then the same coding sets are used for all the blocks in the frame. In this latter case, the coding sets are signaled in the DCTACFRM and DCTACFRM2 fields.

4.1.1.3 Frame-level DCT AC Table Index

DCTACFRM and DCTACFRM2 are variable-length fields that are present in the picture layer if DCTACMBF = 0. The DCTACFRM and DCTACFRM2 fields provide the indices that select the coding sets used to decode the DCT AC coefficients for the Y and Cr/Cb blocks, respectively, in the frame. Table 20 is used to decode the DCTACFRM and DCTACFRM2 fields. Refer to section 4.1.3.4 for a description of AC coefficient decoding.

4.1.1.4 Intra DCT DC Table

DCTDCTAB is a one-bit field that signals which of two Huffman tables is used to decode the DCT DC coefficients in intra-coded blocks. If DCTDCTAB = 0 then the low motion huffman table is used. If DCTDCTAB = 1 then the high motion huffman table is used. Section 5.4 lists the DCT DC Huffman tables.

4.1.1.5 Picture Resolution Index

The RESPIC field in I pictures specifies the scaling factor of the decoded I picture relative to a full resolution frame. The decoded picture may be full resolution or half the original resolution in either the horizontal or vertical dimensions or half resolution in both dimensions. Table 8 shows how the scaling factor is encoded in the RESPIC field.

The following pseudo-code illustrates how the new frame dimensions are calculated if a downsampled resolution is indicated.

X = full resolution horizontal dimension in samples
Y = full resolution vertical dimension in samples
x = new horizontal resolution
y = new vertical resolution
hscale = horizontal scaling factor (0 = full resolution, 1= half resolution)
vscale = vertical scaling factor (0 = full resolution, 1= half resolution)

x = X

```
y = Y
if (hscale == 1)
{
    x = X / 2
    if ((x & 15) != 0)
        x = x + 16 – (x & 15)
}
if (vscale == 1)
{
    y = Y / 2
    if ((y & 15) != 0)
        y = y + 16 – (y & 15)
}
```

If the decoded frame is one of the subsampled resolutions then it must be upsampled to full resolution prior to display. Since this upsampling process is outside the decoding loop, the implementer is free to use any upsampling filter. However, for best results it is advised that the upsampling filter specified in the FILTPIC field be used since the specified filter is designed to match the downsampling filter used to produce the downsampled version of the original frame.

The following section describes the upsampling and downsampling filters.

4.1.1.6 Upsample/Downsample Filter Index

The FILTPIC field is always present in the I picture layer as illustrated in Figure 8. FILTPIC indicates which of the two possible upsampling/downsampling filter pairs is used for resampling a frame. Resampling can occur inside or outside the decoding loop. Out-of-loop resampling occurs when a downsampled frame is upsampled prior to display. In-loop resampling occurs if a resolution change occurs at a P picture. In this case, the reference frame for that P picture must be upsampled or downsampled to match the new resolution. Since out-of-loop upsampling can use any upsampling filter, the FILTPIC index can be ignored. For the Microsoft implementation of the codec, it indicates the filter that will give the optimal upsampling quality. For in-loop resampling of the reference frame, the decoder must use the exact filtering method indicated by the FILTPIC field. The reason that FILTPIC is encoded in the I picture header even though in-loop resampling is never used in I pictures is because FILTPIC is not encoded in P frame picture headers, unless the filter type changes. Therefore, the filter type specified in the I frame FILTPIC field is intended to apply to all subsequent frames until either the filter type changes (FILTFLAG = 1 in the P picture header) or another I picture is decoded.

The following sections describe the upsampling and downsampling process. The following definitions are used for the downsampling/upsampling pseudocode examples:

N_u = number of samples in upsampled (full resolution) line

N_d = number of samples in a downsampled (half resolution) line

$x_u[n]$ = upsampled sample value at position n where n = 0, 1, 2 ... N_u-1

$x_d[n]$ = downsampled sample value at position n where n = 0, 1, 2 ... N_d-1

The term 'line' refers to all the samples in a horizontal row or vertical column in a Y, Cr or Cb component plane. Upsampling or downsampling operations are identical for both rows and columns so the following examples are illustrated using one dimensional line of samples. In cases where both vertical and horizontal upsampling or downsampling is performed, the horizontal lines are resampled first followed by the vertical lines.

For luminance lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution luminance line)
if ((N_d & 15) != 0)
        $N_d = N_d + 16 – (N_d$ & 15)

For chroma lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution chroma line)
if ((N_d & 7) != 0)

$$N_d = N_d + 8 - (N_d \& 7)$$

Downsampling a line must produce the same output as the following pseudocode.

```
if (Nd != (Nu/2))
{
     for (i = Nu; i < Nd*2; i++)
          xu[i] = xu[Nu − 1]
}
downsamplefilter_line(xu[])
for (i = 0; i < Nd; i ++)
     xd[i] = xu[i*2]
```

Upsampling a line must produce the same output as the following pseudocode.

```
for (i = 0; i < Nu; i ++)
{
     xu[i] = xd[i*2]
     xu[i + 1] = 0
}
upsamplefilter_line(xu[])
```

The downsample_filterline() and upsample_filterline() operations are specified in the following sections.

Filter Pair 1 (6-tap downsample / 10-tap upsample)

FILTPIC = 0 indicates that the following upsampling and downsample filtering operations are to be performed when resampling a reference frame

```
AW1 = 70
AW2 = 5
AW3 = -11
```

```
downsamplefilter_line(x[])
{
   y[0] = (((x[0] + x[1]) * AW1 + (x[2] + x[0]) * AW2 + (x[3] + x[1]) * AW3+ RND_DOWN) >> 7)

   for(Int j = 2 j < Nu -2 j += 2) {
      y[j] = (((x[j] + x[j+1]) * AW1 + (x[j-1] + x[j+2]) * AW2 + (x[j-2] + x[j+3]) * AW3 + RND_DOWN) >> 7)
   }


   y[Nu-2] = (((x[Nu-2] + x[Nu-1]) * AW1 + (x[Nu-3] + x[Nu-1]) * AW2 + (x[Nu-4] + x[Nu-2]) * AW3 +
RND_DOWN) >> 7)

   for(j = 0 j < Nu j+=2) {
      x[j] = CLIP(y[j])
      x[j+1] = 0
   }
}
```

RND_DOWN is set to the value 64 when the image is filtered in the horizontal direction, and is set to the value 63 when the image is filtered in the vertical direction.

```
SW1 = 28
SW2 = 6
SW3 = -3
upsamplefilter_line(x[])
{
    y[0] = ((x[0] * SW1 + x[0] * SW2 + x[2] * SW3 + x[4] + RND_UP) >> 5)
    y[1] = ((x[0] * SW1 + x[2] * SW2 + x[0] * SW3 + x[2] + RND_UP) >> 5)
    y[2] = ((x[2] * SW1 + x[0] * SW2 + x[4] * SW3 + x[6] + RND_UP) >> 5)
    y[3] = ((x[2] * SW1 + x[4] * SW2 + x[0] * SW3 + x[0] + RND_UP) >> 5)

    for( j = 4 j < Nu - 4 j += 2) {
        y[j] = ((x[j] * SW1 + x[j-2] * SW2 + x[j+2] * SW3 + x[j+4] + RND_UP) >> 5)
        y[j+1] = ((x[j] * SW1 + x[j+2] * SW2 + x[j-2] * SW3 + x[j-4] + RND_UP) >> 5)
    }

    y[Nu-4] = ((x[Nu-4] * SW1 + x[Nu-6] * SW2 + x[Nu-2] * SW3 + x[Nu-2] + RND_UP) >> 5)
    y[Nu-3] = ((x[Nu-4] * SW1 + x[Nu-2] * SW2 + x[Nu-6] * SW3 + x[Nu-8] + RND_UP) >> 5)
    y[Nu-2] = ((x[Nu-2] * SW1 + x[Nu-4] * SW2 + x[Nu-2] * SW3 + x[Nu-4] + RND_UP) >> 5)
    y[Nu-1] = ((x[Nu-2] * SW1 + x[Nu-2] * SW2 + x[Nu-4] * SW3 + x[Nu-6] + RND_UP) >> 5)

    for(j = 0 j < Nu j++)
        x[j] = CLIP(y[j])
}
```

RND_UP and is set to the value 15 when the image is filtered in the horizontal direction, and is set to the value 16 when the image is filtered in the vertical direction.

Filter Pair 2 (5-tap downsample / 3-tap upsample)

FILTPIC = 1 indicates that the following upsampling and downsample filtering operations are to be performed when resampling a reference frame

```
downsamplefilter_line(x[])
{
    for (j = 1; j < Nu-1; j += 2)
        x[j] =((x[j]<<8) - ((x[j-1]+x[j+1])<<7));
    x[Nu-1] = (x[Nu-1]<<8) - (x[Nu-2]<<8);

    x[0] = (x[0]<<8) + (x[1]>>1);
    for (j = 2; j < Nu; j += 2)
        x[j] = (x[j]<<8) + ((x[j-1]+x[j+1])>>2);

    for (j = 0; j < Nu; j += 2) {
        x[j] = ((x[j]+128)>>8);
        x[j] = CLIP(x[j]);
        x[j+1] = 0;
```

```
        }
    }


upsamplefilter_line(x[])
{
    for (j = 1; j < Nu-1; j += 2)
        x[j] =((x[j-1]+x[j+1])>>1);
    x[Nu-1] = x[Nu-2];
}
```

4.1.1.7   Preprocess Frame - I Frame (PREPROCFRM)

The PREPROCFRM is only signaled when PREPROC is signaled at the sequence level.

When PREPROCFRM is signaled for the current I Frame, we have to scale up the current decoded frame prior to display while keeping the actual reconstructed frame for the possibility of use in future motion compensation. Let Y, U, V denote the YUV planes of the output frame. We scale them up according to the following formula:

$$Y_p[n] = (Y[n] - 128) << 1 + 128;$$

$$U_p[n] = (U[n] - 128) << 1 + 128;$$

$$V_p[n] = (V[n] - 128) << 1 + 128;$$

4.1.2 Macroblock Layer Decode

Figure 2 shows how the frame is composed of macroblocks. The macroblocks are coded in raster scan order form left to right. Figure 15 shows the elements that make up the I picture macroblock layer.

4.1.2.1   Coded Block Pattern

The coded block pattern specifies which of the six blocks that make up the macroblock have AC coefficient information coded within the bitstream. The coded block pattern is derived from the six-bit value obtained from decoding the variable-length CBPCY field in the macroblock header (the Huffman table used to decode CBPCY is listed in section 5.2). The coded block pattern (*cbpcy*) is derived from the six-bit value decoded from the CBPCY field (*decoded_cbpcy*) as follows:

$cbpcy = decoded\_cbpy \wedge predicted\_cbpy$

where *predicted_cbpy* is calculated as follows:

$predicted\_cbpy = (predicted\_Y1 << 5) \mid (predicted\_Y2 << 4) \mid (predicted\_Y3 << 3) \mid (predicted\_Y4 << 2)$

where *predicted_Y1 .. predicted_Y4* are each one-bit values calculated as follows:

predicted_Y1 =
> *L2*, if *LT4* equals *T3*
> *T3* otherwise

predicted_Y2 =
> *predicted_Y1*, if *T3* equals *T4*
> *T4* otherwise

predicted_Y3 =
> *L4*, if *L2* equals *predicted_Y1*
> *predicted_Y1* otherwise

predicted_Y4 =
> *predicted_Y3*, if *predicted_Y1* equals *predicted_Y2*
> *predicted_Y2* otherwise

L1, L2, L3, L4, LT4, T1, T2, T3 and T4 are one-bit values representing the coded status of the neighboring luminance blocks as illustrated in Figure 29. The figure shows the four luminance blocks which make up the current

macroblock outlined in a heavy border along with blocks from the neighboring macroblocks. The values of T1, T2, etc indicate whether the corresponding block was coded or not. For example, if L1 = 1 then block Y1 in the macroblock to the immediate left of the current macroblock was coded. If L1 = 0 then the block was not coded.

		T1	T2
	LT4	T3	T4
L1	L2	Current Macroblock (Y Blocks)	
L3	L4		
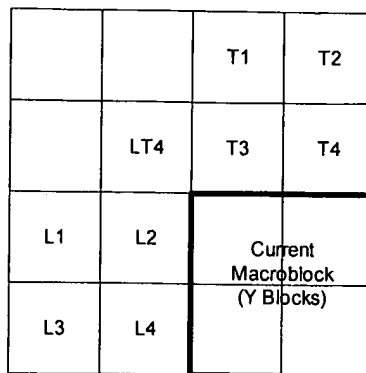
Figure 29: CBP encoding using neighboring blocks

The six-bit coded block pattern (*cbpcy*) specifies which of the six blocks that make up the macroblock have at least one non-zero AC coefficient coded in the block layer bitstream. The bit positions in the six-bit coded block pattern field correspond to the six blocks as shown in Table 38 (bit position 0 is the rightmost bit):

Table 38: Coded block pattern bit position

	Coded Block Pattern Bit Position					
	5	4	3	2	1	0
Block	Y1	Y2	Y3	Y4	Cr	Cb

A bit value of 1 in the coded block pattern indicates that the corresponding block has at least one non-zero AC coefficient coded in the block layer bitstream. A value of zero indicates that there are no AC coefficients coded in the block layer bitstream.

4.1.2.2 AC Prediction Flag

The ACPRED field in the macroblock header is a one-bit field that specifies whether AC prediction is used to decode the AC coefficients for all the blocks in the macroblock. Section 4.1.3.7 describes the AC prediction process. If ACPRED is 1 then AC prediction is used, otherwise it is not used.

4.1.2.3 Macroblock-level DCT AC Table Index

The DCTTAB field is present in the macroblock header if the DCTACMBF field in the picture header indicates that macroblock-level AC coding set selection is enabled (described in section 4.1.1.2). DCTAB is a variable-length code that specifies a coding set index. The index is used to select the coding set used to decode the AC coefficient information. Refer to section 4.1.3.4 for a description of AC coefficient decoding. Table 20 shows the code-table used to decode the DCTTAB field.

4.1.3 Block Decode

Figure 2 illustrates how each macroblock is made up of 6 blocks. As the figure shows, the 4 blocks that make up the Y component of the macroblock are coded first followed by the Cr and Cb blocks. This section describes the process used to reconstruct the blocks.

Figure 3 shows the forward intra-coding steps used to encode the 8x8 pixel blocks. Figure 30 shows the inverse process used to reconstruct the 8x8 blocks.
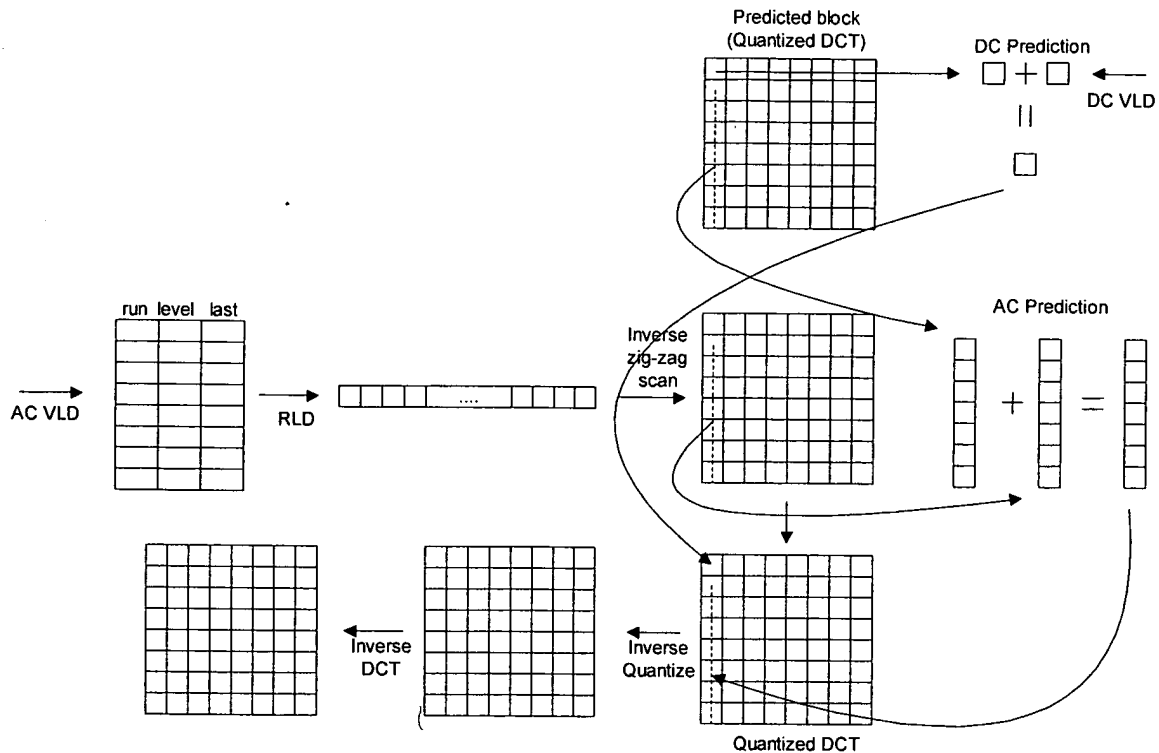
Figure 30: Intra block reconstruction

As Figure 30 shows, the DC and AC DCT coefficients are coded using separate techniques. The DC coefficient is coded differentially. An optional differential coding of the left or top AC coefficients can be used. The following sections describe the process for reconstructing intra blocks in I pictures

4.1.3.1 DC Differential Bitstream Decode

The DC coefficient is coded differentially with respect to an already-decoded DC coefficient neighbor. This section describes the process used to decode the bitstream to obtain the DC differential.

Figure 24 shows the bitstream elements used to encode the DC differential. DCCOEF is decoded using one of two VLC code tables. The table is specified by the DCTDCTAB field in the picture header (see section 4.1.1.4). Based on the value of DCTDCTAB, one of the two Huffman tables listed in section 5.4 is used to decode DCCOEF. This will yield either:

1) Zero, or

2) the absolute value of the DC differential, or

3) The escape code.

If DCOEF decodes to zero, the value of the DC differential is also zero. Other wise, further decoding is necessary to determine the value of DC differential. If the DCCOEF decodes to the escape code, the absolute value of the DC differential is encoded in the DCCOEFESC field (section 3.2.3.3). The size of the DCCOEFESC field can be 8, 9 or 10 bits depending on the quantization step size of the block. The sign of the DC differential is obtained from the DCSIGN field (section 3.2.3.4).

The following pseudo-code illustrates the DC differential decoding process:

```
DCDifferential = vlc_decode()
if(DCDifferential != 0) {
    if(DCDifferential == ESCAPECODE) {
        if(QUANT == 1)
            DCDifferential = flc_decode(10);
```

```
        else if(QUANT == 2)
          DCDifferential = flc_decode(9);
        else // QUANT is > 2
          DCDifferential = flc_decode(8);
    }
  else {  // DCDifferential is not ESCAPECODE
    if(QUANT == 1)
      DCDifferential = DCDifferential*4 + flc_decode(2) - 3;
    else if(QUANT == 2)
      DCDifferential = DCDifferential*2 + flc_decode(1) - 1;
    }
  DCSign = flc_decode(1)
  if (DCSign == 1)
    DCDifferential = -DCDifferential
}
```

Figure 31: DC Differential Decoding Pseudo-code

4.1.3.2   DC Predictor

A	B
C	Current Block
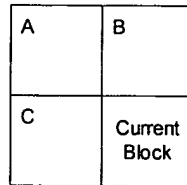
Figure 32: DC predictor candidates

The quantized DC value for the current block is obtained by adding the DC predictor to the DC differential obtained as described in section 4.1.3.1. The DC predictor is obtained from one of the previously decoded adjacent blocks. Figure 32 above shows the current block and the candidate predictors from the adjacent blocks. The values A, B and C represent the quantized DC values for the top-left, top and left adjacent blocks respectively.

In the following cases there are no adjacent blocks:

1) The current block is in the first block row of the frame. In this case there are no A or B (and possibly C) blocks

2) The current block is in the first block column in the frame. In this case there are no A and C (and possibly B) blocks.

For these cases the DC predictor is set to:

DCPredictor = (1024 + (DCStepSize >> 1)) / DCStepSize

Refer to section 4.1.3.3 for a description of how to compute DCStepSize.

A prediction direction is formed based on the values of A, B and C and either the B or C predictor is chosen. The prediction direction is calculated as follows:

If the absolute value of (A - B) is less than or equal to the absolute value of (A − C) then the prediction is made from the left (C is the predictor). Otherwise the prediction is made from the top (B is the predictor). In pseudo-code, the process is as follows:

```
if (|A − B| <= |A − C|)
{
    PredDirection = left;
    DCPredictor = C;
}
```

```
else
{
    PredDirection = top;
    DCPredictor = B;
}
```

Figure 33: Prediction selection pseudo-code

The quantized DC coefficient is then calculated by adding the DC differential and the DC predictor as follows:

$$DCCoeffQ = DCPredictor + DCDifferential$$

4.1.3.3   DC Inverse-quantization

The quantized DC coefficient is reconstructed by performing the following de-quantization operation:

$$DCCoefficient = DCCoeffQ * DCStepSize$$

The value of DCStepSize is based on the value of PQUANT (obtained in the picture header and described in section 3.2.1.7) as follows:

For PQUANT equal to 1 or 2:

$$DCStepSize = 2 * PQUANT$$

For PQUANT equal to 3 or 4:

$$DCStepSize = 8$$

For PQUANT greater than or equal to 5:

$$DCStepSize = PQUANT / 2 + 6$$

4.1.3.4   AC Coefficient Bitstream Decode

The non-zero quantized AC coefficients are coded using a run-level method. A set of tables and constants are used to decode the *run*, *level* and *last-flag* values. For descriptive purposes, the set of tables and constants is called an **AC coding set**. Following is a description of the tables and constants that make up an AC coding set.

Tables The first step in reconstructing the AC DCT coefficients is to decode the bitstream to obtain the *run*, *level* and *last-flag* triplets that represent the location and quantized level for each non-zero AC coefficient.

Huffman table (HuffTable):   The code table used to decode the ACCOEF1 and ACCOEF2 variable-length encoded fields.

Run table (RunTable): The table of *run* values indexed by the value decoded in the ACCOEF1 or ACCOEF2 fields

Level table (LevelTable): The table of level values indexed by the value decoded in the ACCOEF1 or ACCOEF2 fields.

Not-last delta run table (NotLastDeltaRunTable): The table of delta *run* values indexed by the *level* value as illustrated in pseudo-code of Figure 34. Used in escape coding mode 2.

Last delta run table (LastDeltaRunTable): The table of delta *run* values indexed by the *level* value as illustrated in pseudo-code of Figure 34. Used in escape coding mode 2.

Not-last delta level table (NotLastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 34. Used in escape coding mode 1.

Last delta level table (LastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 34. Used in escape coding mode 1.

Constants

Start index of last coefficient (StartIndexOfLast): The HuffTable encodes index values from 0 to N. The index values are used to obtain the run and level values from RunTable and LevelTable respectively. The first (StartIndexOfLast-1) of these index values correspond to run, level pairs that are not the last pair in the block. The next StartIndexOfLast to N-1 index values correspond to run, level pairs that are the last pair in the block. The last value, N, is the Escape Index (see next).

Escape Index (EscapeIndex): The last in the set of indices encoded by HuffTable. See the description above and the pseudo-code of Figure 34 for a description of how this constant is used.

The following pseudo-code illustrates how the tables and constants are used to decode a run, level and last-flag triplet.

```
last_flag = 0;
index = vlc_decode();    ## Use HuffTable to decode VLC codeword (ACCOEF1)
If (index != EscapeIndex)
{
     run = RunTable[index];
     level = LevelTable[index];
     sign = get_bits(1);
     if (sign == 1)
          level = -level;
     if (index >= StartIndexOfLast)
          last_flag = 1;
}
else
{
     escape_mode = vlc_decode();      ## Use HuffTable to decode ESCMODE field
     if (escape_mode == mode1)
     {
          index = vlc_decode();    ## Use HuffTable to decode VLC codeword (ACCOEF2)
          run = RunTable[index];
          level = LevelTable[index];
          if (index >= StartIndexOfLast)
               last_flag = 1;
          if (last_flag == 0)
               level = level + NotLastDeltaLevelTable[run];
          else
               level = level + LastDeltaLevelTable[run];
          sign = get_bits(1);
          if (sign == 1)
               level = -level;

     }
     else if (escape_mode == mode2)
     {
          index = vlc_decode();    ## Use HuffTable to decode VLC codeword (ACCOEF2)
          run = RunTable[index];
          level = LevelTable[index];
          if (index >= StartIndexOfLast)
               last_flag = 1;
          if (last_flag == 0)
               run = run + NotLastDeltaRunTable[level];
          else
```

```
                run = run + LastDeltaLevelTable[level];
        sign = get_bits(1);
        if (sign == 1)
                level = -level;
}
else ## escape_mode == mode3 (fixed-length encoding)
{
        if (first_mode3_in_frame == 1)
        {
                first_mode3_in_frame = 0;
                level_code_size = vlc_decode();   ## Use Table 26 or Table 27 to decode
                run_code_size = 3 + get_bits(2);
        }
        run = get_bits(run_code_size);
        sign = get_bits(1);
        level = get_bits(level_code_size);
        if (sign == 1)
                level = -level;
}
}
```

Figure 34: Coefficient decode pseudo-code

The process illustrated in Figure 34 above for decoding the non-zero AC is repeated until *last_flag* = 1. This flag indicates the last non-zero coefficient in the block.

To improve coding efficiency, there are six AC coding sets. The six coding sets are divided into two groups of three, nominally called intra and inter coding sets. For Y blocks, one of the three intra coding sets is used. For Cr and Cb blocks one of the three inter coding sets is used. Section 5.5 lists the tables that make up each coding set. The particular set used to decode a block is signaled by an index value in either the picture or macroblock header. The following table shows how the index corresponds to the coding set for Y and Cr/Cb blocks.

To improve coding efficiency, there are eight AC coding sets. The eight coding sets are divided into two groups of four, nominally called intra and inter coding sets. For Y blocks, one of the four intra coding sets is used. For Cr and Cb blocks one of the four inter coding sets is used. Section 5.5 lists the tables that make up each coding set. The particular set used to decode a block is signaled by an index value in either the picture or macroblock header. The following two tables shows how the index corresponds to the coding set for Y and Cr/Cb blocks. As the tables show, if the value of PQINDEX (see section 3.2.1.7) is less than or equal to 7 then the high rate coding set is used for index 0. If PQINDEX is greater than 7 then the low motion coding set is used for index 0.

Table 39: Coding Set Correspondance for PQINDEX <= 7

Y blocks		Cr and Cb blocks	
Index	Table	Index	Table
0	High Rate Intra	0	HighRate Inter
1	High Motion Intra	1	High Motion Inter
2	MPEG-4 Intra	2	MPEG-4 Inter

Table 40: Coding Set Correspondance for PQINDEX > 7

Y blocks		Cr and Cb blocks	
Index	Table	Index	Table
0	Low Motion Intra	0	Low Motion Inter
1	High Motion Intra	1	High Motion Inter

2	MPEG-4 Intra	2	MPEG-4 Inter

Whether the index is specified at the picture or macroblock level is signaled by the value of DCTACMBF specified in the picture layer (see section 4.1.1.2 for a description). If the index is specified in the picture header, then the value decoded from the DCTACFRM2 field is used as the coding set index for Y blocks and the value decoded from the DCTACFRM field is used as the coding set index for Cr and Cb blocks. If the index is signaled at the macroblock level then the value decoded from the DCTTAB field is used as the index for both inter and intra sets.

4.1.3.5 AC Run-level Decode

The ordered run and level pairs obtained as described in section 4.1.3.4 above are used to form an array of 63 elements by employing a run-level decode process as illustrated in the pseudo-code of Figure 35.

```
array[63]      ## 63 element array
curr_position = 0;
do {
    decode_symbol(&run, &level, &last_flag); ## decode the bitstream as described in Figure 34 to
                                             ## obtain run, level and last_flag values for coefficient
    array[curr_position + run] = level;
    curr_postion = curr_postion + run + 1;
} while (last_flag != 1)
```

Figure 35: Run-level decode pseudo-code

4.1.3.6 Zig-zag Scan of AC Coefficients

Decoding the run-level pairs as described in section 4.1.3.5 produces a one-dimensional array of 63 quantized AC coefficients. The elements in the array are scanned out into an 8x8 two-dimension array in preparation for the IDCT. Figure 36 shows the elements in an 8x8 array labeled in raster scan order from 0 to 63. The DC coefficient is in position 0. A mapping array is used to scan out the remaining 63 AC coefficients in the one-dimensional array to the 8x8 array. As an example, Figure 38 shows the mapping array used to produce the one-dimensional to two-dimensional scan out pattern shown in Figure 37

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figure 36: 8x8 array with positions labeled
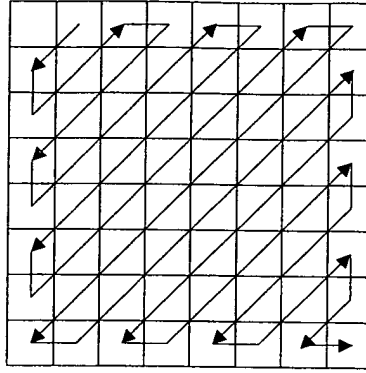
Figure 37: Example zig-zag scanning pattern

| 1 | 8 | 16 | 9 | 2 | 3 | 10 | 17 | 24 | 32 | 25 | 18 | 11 | 4 | 5 | 12 | 19 | 26 | 33 | 40 | 48 | 41 | 34 | 27 | 20 | 13 | 6 | 7 |

| 14 | 21 | 28 | 35 | 42 | 49 | 56 | 57 | 50 | 43 | 36 | 29 | 22 | 15 | 23 | 30 | 37 | 44 | 51 | 58 | 59 | 52 | 45 | 38 | 31 | 39 | 46 | 53 |

| 60 | 61 | 54 | 47 | 55 | 62 | 63 |

Figure 38: Zig-zag scan mapping array

One of three scan arrays is used to scan out the one-dimensional array depending on the AC prediction status for the block (see section 4.1.3.7 for description of AC prediction). Table 41 shows how the AC prediction status determines which scan array is used.

Table 41: Scan Array Selection

AC Prediction	AC Scan Array
Top prediction	Horizontal scan
Left prediction	Vertical scan
No prediction	Normal scan

The tables for the horizontal, vertical and normal scan arrays are listed in section 5.6.1.

4.1.3.7 AC Prediction

If the ACPRED field in the macroblock layer specifies that AC prediction is used for the blocks then the top row or left column of AC coefficients in the decoded block are treated as differential values from the coefficients in the corresponding row or column in a predicted block. The predicted block is either the block immediately above or to the left of the current block. For each block, the direction chosen for the DC predictor is used for the AC predictor (see section 4.1.3.2). Figure 39 shows that for top prediction the first row of AC coefficients in the block immediately above is used as the predictor for the first row of AC coefficients in the current block. For left prediction the first column of AC coefficients in the block to the immediate left is used as the predictor for the first column of AC coefficients in the current block.
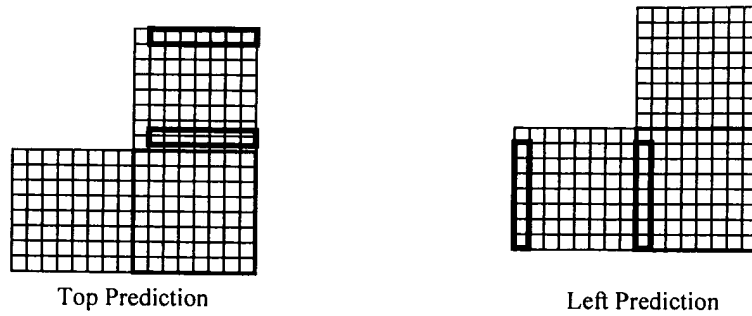
Figure 39: AC prediction candidates

If a block does not exist in the predicted direction then the predicted values for all 7 coefficients are set to zero. For example, if the prediction is up but the block is in the top row then there is no adjacent block in the up direction.

The AC coefficients in the predicted row or column are added to the corresponding decoded AC coefficients in the current block to produce the fully reconstructed quantized DCT coefficient block.

4.1.3.8 Inverse AC Coefficient Quantization

Depending on whether the 3-QP or 5-QP deadzone quantizer is used (see section 3.2.1.7), the non-zero quantized AC coefficients reconstructed as described in the sections above are inverse quantized according to the following formula:

dequant_coeff = *quant_coeff* * *double_quant* (if 3-QP deadzone quantizer), or

dequant_coeff = *quant_coeff* * *double_quant* + sign(*quant_coeff*) * *quant_scale* (if 5-QP deadzone quantizer)

where:

> *quant_coeff* is the quantized coefficient
>
> *dequant_coeff* is the inverse quantized coefficient
>
> *double_quant* = 2 * PQUANT + HalfStep
>
> *quant_scale* = PQUANT

PQUANT is encoded in the picture layer as described in section 3.2.1.7. HalfStep is encoded in the picture layer as described in section 3.2.1.8.

4.1.3.9 Inverse DCT

After reconstruction of the DCT coefficients, the resulting 8 × 8 blocks are processed by a separable two-dimensional inverse discrete cosine transform of size 8 by 8. The inverse transform output ranges from 0 to +255 after clipping to be represented with 8 bits. The transfer function of the inverse transform is given by:

$$f(x,y) = 1/4 \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v)\cos[\pi(2x+1)u/16]\cos[\pi(2y+1)v/16]$$

with $u, v, x, y = 0, 1, 2, \ldots, 7$

where x,y = spatial coordinates in the pixel domain,

> u,v = coordinates in the transform domain,
>
> $C(u) = 1/\sqrt{2}$ for $u = 0$, otherwise 1,
>
> $C(v) = 1/\sqrt{2}$ for $v = 0$, otherwise 1.

NOTE – Within the block being transformed, $x = 0$ and $y = 0$ refer to the pixel nearest the left and top edges of the picture respectively.

See section 4.11 regarding IDCT conformance.

4.2 Interlace I Frame Decoding

The following sections describe the process for decoding Interlace I pictures.

4.2.1 Interlace I Picture Layer Decode

Figure 9 shows the elements that make up the Interlace I picture layer header. Most of the elements are the same as the baseline I picture layer. The following sections provide extra detail for some of the elements that are unique to Interlace I picture.

4.2.1.1 Picture Resolution Index

The RESPIC2 field in I pictures specifies the scaling factor of the decoded I picture relative to a full resolution frame. For Interlace pictures, the decoded picture may be full resolution or half the original resolution in the horizontal dimensions. Table 9 shows the how the picture resolution is coded in the picture layer.

4.2.1.2 Upsample/Downsample Filter Index

Same as Progressive I Frame as describe section 4.1.1.6.

4.2.1.3 Interlaced Field/Frame Decoding

If the sequence layer field INTERLACE = 1 then a picture layer field INTRLCF is present in the bitstream. INTRLCF is a 1-bit field that indicates the mode used to code the macroblocks in that frame. If INTRLCF = 0 then all macroblocks in the frame are coded in frame mode. If INTRLCF = 1 then the macroblocks may be coded in field or frame mode and the INTRLCMB field is present in the picture layer. INTRLCMB is a bitplane coded field that indicates the field/frame coding status for each macroblock in the picture. The decoded bitplane represents the interlaced status for each macroblock as a field of 1-bit values in raster scan order from upper left to lower right. Refer to section 4.10 for a description of the bitplane coding. A value of 0 indicates that the corresponding macroblock is coded in frame mode. A value of 1 indicates that the corresponding macroblock is coded in field mode.

4.2.2 Macroblock Layer Decode

The macroblocks are coded in raster scan order form left to right. Each macroblock can be either frame or field encoded as indicated by INTRLCMB. Figure 16 shows the elements that make up the intra frame MB layer and Figure 17 shows the elements that make up the intra field MB layer. Each macroblock is composed of 4 Y blocks Y_0, Y_1, Y_2, Y_3, and 2 U blocks U_0, U_1, and 2 V blocks V_0, V_1. Each one of the Y blocks (Y_0, Y_1, Y_2, Y_3) are 8x8 while the chrominance blocks U_0, U_1, V_0, V_1 are 4x8. For frame MB, we send the blocks in the following order Y_0, Y_1, Y_2, Y_3, U_0, U_1, V_0, V_1. For field MB, we first rearrange the MB according to field and then send the blocks in the following order Y_0, Y_1, U_0, V_0, Y_2, Y_3, U_1, V_1 as described in Section 2.2

4.2.2.1 Coded Block Pattern

The coded block pattern consists of 6 bits, one bit for each the four Y blocks, one bit for both U blocks, and one bit for both V blocks. It specifies whether the block(s) have AC coefficients or not and it is encoded the same way as baseline I frame. For the Y blocks, a bit value of 1 in the coded block pattern indicates that the corresponding block has at least one non-zero AC coefficient coded in the block layer bitstream. A value of zero indicates that there are no AC coefficients coded in the block layer bitstream. For the U or V blocks, a bit value of 1 in the coded block pattern indicates that at least one of the two blocks have non-zero AC coefficients coded in the block layer bitstream.

4.2.2.2 Subblock Pattern U,V

When CBP indicates there are nonzero U blocks, we need to specify which block(s) have nonzero AC coefficients. This is done by sending a subblock pattern that will indicate whether U_0, U_1, or both have nonzero AC coefficients. The two blocks U_0, U_1 are grouped together and encoded using the VLC table shown in Table 42 where an "X" denote that there are nonzero AC coefficients.

Similarly, we need to specify the subblock pattern for the V blocks when CBP indicates there are nonzero V blocks. The V blocks are grouped together and coded with the same table as the U blocks.

Table 42: Subblock pattern U,V code-table

SUBBLKPATU VLC		
	U_0	U_1
0	X	X
10		X
11	X	

4.2.2.3 , AC Prediction MB, AC Prediction Top Field, Bottom Field

The ACPREDMB flag is only present for frame encoded MB. It is a one bit value specifying whether AC prediction is used for all the blocks (i.e.Y_0, Y_1, Y_2, Y_3, U_0, U_1, V_0, V_1) in the Macroblock.

The ACPREDTFIELD and ACPREDBFIELD flags are only present in field encoded MB. Both flags are one bit value specifying whether AC prediction is used for the top or the bottom field. Thus if ACPREDTFIELD is on, then AC prediction is on for blocks Y_0, Y_1, U_0, V_0. Similarly, if ACPREDBFIELD is on, then AC prediction is on for blocksY_2, Y_3, U_1, V_1.

4.2.3 Block Decode

Figure 16 shows the elements that make up the intra frame MB layer and Figure 17 shows the elements that make up the intra field MB layer. Each macroblock is composed of 4 Y blocks Y_0, Y_1, Y_2, Y_3, and 2 U blocks U_0, U_1, and 2 V blocks V_0, V_1. Each one of the Y blocks (Y_0, Y_1, Y_2, Y_3) are 8x8 while the chrominance blocks U_0, U_1, V_0, V_1 are 4x8. For frame MB, we send the blocks in the following order Y_0, Y_1, Y_2, Y_3, U_0, U_1, V_0, V_1. For field MB, we send the blocks in the following order Y_0, Y_1, U_0, V_0, Y_2, Y_3, U_1, V_1.

The decoding process for the luminance blocks Y_0, Y_1, Y_2, Y_3, and chrominance blocks U_0, U_1, V_0, V_1 are the same for frame encoded MB and field encoded MB. Figure 40 shows the block decoding process for a luminance block and Figure 41 shows the block decoding process for a chrominace block. For both the luminance and chrominance blocks, the DC coefficients are encoded differentially using neighboring block's DC coefficients. In addition, the first row's or the first column's AC coefficients might be differentially coded as signaled by the ACPREDMB, ACPREDTFIELD, or ACPREDBFIELD flags. We note that for a chrominance block, if row AC prediction is chosen, then the four coefficients of the first row are differentially coded.
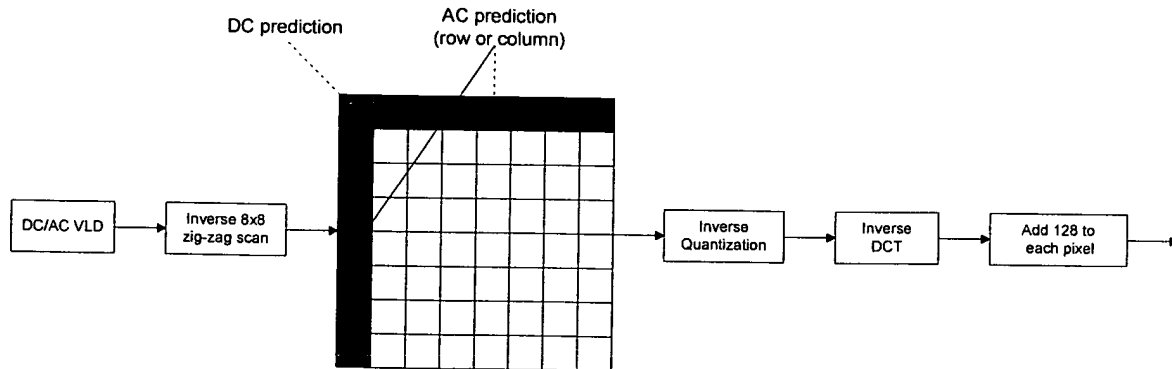
Figure 40: Intra Luminance (8x8) Block Decode
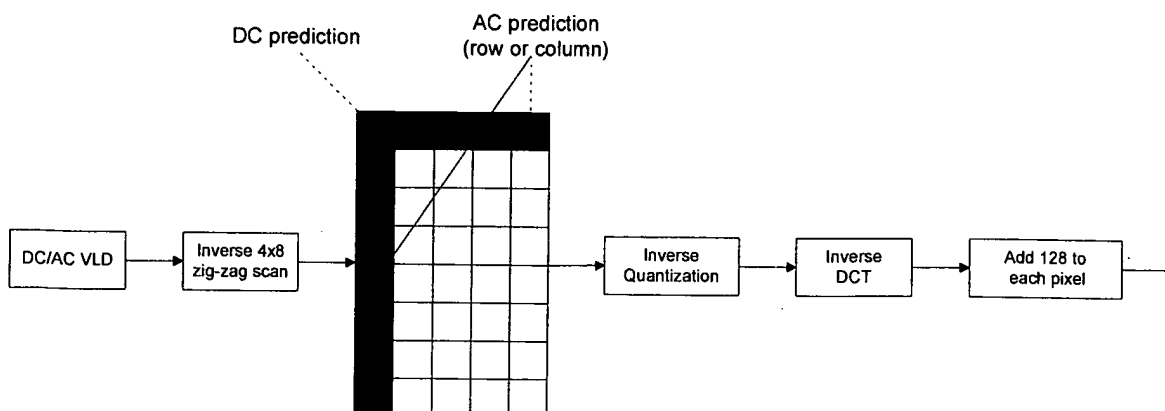
Figure 41: Intra Chrominance (4x8) Block Decode

4.2.3.1　DC Differential Bitstream Decode

The DC differential is obtained after DC prediction. The process used to decode the DC differential from the bitstream is the same one used in baseline I frame as described in section 4.1.3.1.

4.2.3.2　DC Predictor
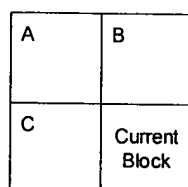
Figure 42: DC predictor candidates

The quantized DC value for the current block is obtained by adding the DC predictor to the DC differential obtained as described in section 4.2.3.1. The DC predictor is obtained from one of the previously decoded adjacent blocks. Figure 42 above shows the current block and the candidate predictors from the adjacent blocks. The values A, B and C represent the quantized DC values prior to the addition of 128 for the top-left, top and left adjacent blocks respectively.

The adjacent blocks A, B, C are considered missing if they are outside the picture boundary or if the blocks are not intra coded (the last provision is for intra blocks in Interlace p pictures).

If all three blocks A, B, and C are present, then a prediction direction is formed based on the values of A, B and C and either the B or C predictor is chosen. The prediction direction is calculated the same way as the baseline I frame as shown in Figure 33, section 4.1.3.2.

If an adjacent block is missing, then the following rules applies:

- If block C is missing and block B is not, then use block B as the predictor.

- If block B is missing and block C is not, then use block C as the predictor.

- If both block B and block C are missing, then no predictor is used (remember we are working with quantized coefficients prior to the addition of 128).

- If block A is missing and B, C are present, then we choose block B if the DC predictor for block C is smaller than the DC predictor for block B, otherwise, we choose block C.

In addition, the DC predictor is scaled (see section 4.2.3.8) if MQUANT mode is on.

4.2.3.3　DC Inverse-quantization

The procedure is the same as baseline I frame picture (see section 4.1.3.3) with the exception that each macroblock might have different quantization stepsize specified by MQUANT.

4.2.3.4 AC Coefficient Bitstream Decode

The procedure is the same as baseline I frame picture (see section 4.1.3.4).

4.2.3.5 AC Run-level Decode

The procedure is the same as baseline I frame picture (see section 4.1.3.5).

4.2.3.6 Zig-zag Scan of AC Coefficients

The one dimensional array of quantized coefficients produced in the run-level decoding process described above are scanned out into a two-dimensional array in preparation for the IDCT. The process is similar to that described in section 4.1.3.6. The difference is that the chrominance blocks are 4x8 and uses the zig-zag scan array shown in Table 138 of section 5.6.2. The luminance blocks are 8x8 and still uses the same zig-zag scan array as the baseline I frame shown in Table 133 of section 5.6.1.

4.2.3.7 AC Prediction

If AC prediction is turned on for the current block, then the AC coefficients on either the top row or the left column might be differentially encoded. The decision for the direction is based on the DC predictor. There are three cases, DC is predicted from the left block, the top block, or not predicted.

- If DC is predicted from the top, then the top row of the current block is differentially coded.

- If DC is predicted from the left, then the left column of the current block is differentially coded.

- If DC is not predicted, then the AC coefficients are not differentially coded.

The AC coefficients in the predicted row or column are added to the corresponding decoded AC coefficients (prior to adding 128) in the current block to produce the fully reconstructed quantized DCT coefficient block. In addition, the DC predictor is scaled (see section 4.2.3.8) if MQUANT mode is on.

4.2.3.8 DC/AC Prediction Coefficient Scaling

For DC and AC prediction, the coefficients in the predicted blocks are scaled if the macroblocks quantizers are different than that of the current block. The coefficients are scaled as follows:

$$\overline{DC}_p = (DC_p * DCSTEP_p * DQScale[DCSTEP_c] + 0x20000) >> 18,$$

$$\overline{AC}_p = (AC_p * STEP_p * DQScale[STEP_c] + 0x20000) >> 18$$

where

$\overline{DC}_p$ is the scaled DC coefficient in the predictor block

DC_p is the original DC coefficient in the predictor block

$DCSTEP_p$ is the DC quantizer step size in the predictor block

$DCSTEP_c$ is the DC quantizer step size in the current block

$\overline{AC}_p$ is the scaled AC coefficient in the predictor block

AC_p is the original AC coefficient in the predictor block

$STEP_p$ is the quantizer step size in the predictor block

$STEP_c$ is the quantizer step size in the current block

$DQScale$ is an integer look up table with inputs from 1 to 31.

Table 43: DQScale

Index	DQScale[Index]
1	262144
2	131072
3	87381
4	65536
5	52429
6	43691
7	37449
8	32768
9	29127
10	26214
11	23831
12	21845
13	20165
14	18725
15	17476
16	16384
17	15420
18	14564
19	13797
20	13107
21	12483
22	11916
23	11398
24	10923
25	10486
26	10082
27	9709
28	9362
29	9039
30	8738
31	8456

4.2.3.9 Inverse AC Coefficient Quantization

The procedure is the same as baseline I frame picture (see section 4.1.3.8) with the exception that each macroblock might have different quantization stepsize specified by MQUANT.

4.2.3.10 Inverse DCT

After reconstruction of the DCT coefficients, the luminance blocks is using 8x8 IDCT and the chrominance blocks are using 4x8 IDCT.

4.2.3.11 Compensate Average Value (Add 128)

After IDCT, each pixel in the block will be added with 128 and clipped to be between 0 and 255.

4.3 X8INTRA I Frame Decoding

The following sections describe the process for decoding X8INTRA I pictures. This procedure is used when the X8IF flag is set.

> *4/3/2002: X8INTRA is disabled for the Main and Simple profiles in WMV9. It is described in this document only for completeness (since it is used in WMV8) and future use in the Complex profile of WMV9. The Complex profile of WMV9 is purely for technology demonstration, and is not intended for use by content providers and hardware partners.*

4.3.1 X8INTRA I Picture Layer Decode

Note that the RESPIC and FILTPIC fields in Figure 8 control the current picture resolution and resampling filters. Sections 4.1.1.5 and 4.1.1.6 describe the resampling process.

The syntax of I frames (i.e. I pictures) is made up of three layers: picture, row and block. No information is coded at the macroblock level. The picture-level syntax is shown in Figure 43.
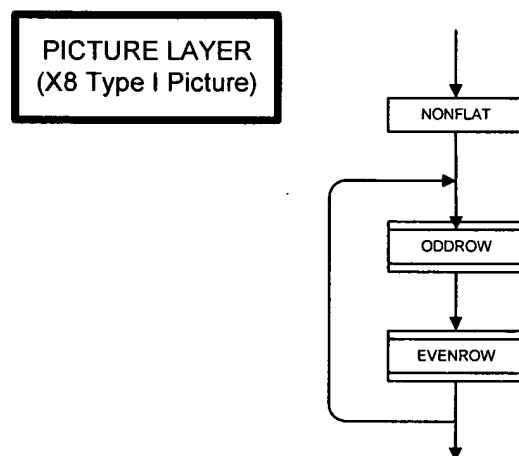
Figure 43: Frame-level syntax for X8 type I picture

4.3.1.1 Non-flat Quantization Flag (NONFLAT) (1 bit)

At the beginning of an X8 type intra frame, a single bit of information, NONFLAT, is sent. This bit, when set, indicates that the dequantization step in a particular nonzero transform coefficient in a block varies according to the index of the coefficient in the relevant scan order. When NONFLAT = 0, all dequantization steps in the frame are identical. This process is explained in Section 4.3.6.1.

69
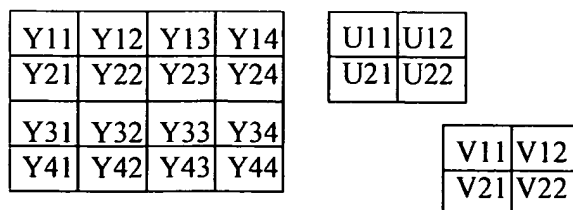
Row 1: [Y11 Y12 Y13 Y14]

Row 2: [Y21 Y22 U11 V11 Y23 Y24 U12 V12]

Row 3: [Y31 Y32 Y33 Y34]

Row 4: [Y41 Y42 U21 V21 Y43 Y44 U22 V22]

Figure 44:Ordering of entropy coded information in a frame

4.3.2 X8INTRA Row Layer Decode
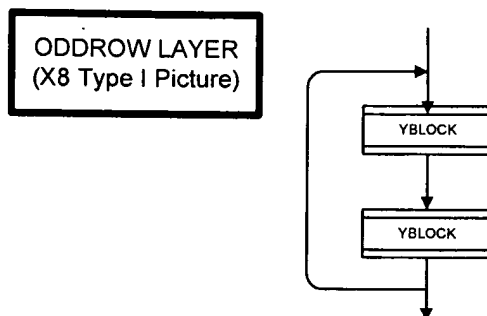
Figure 45: Syntax of odd rows in an X8 intra frame
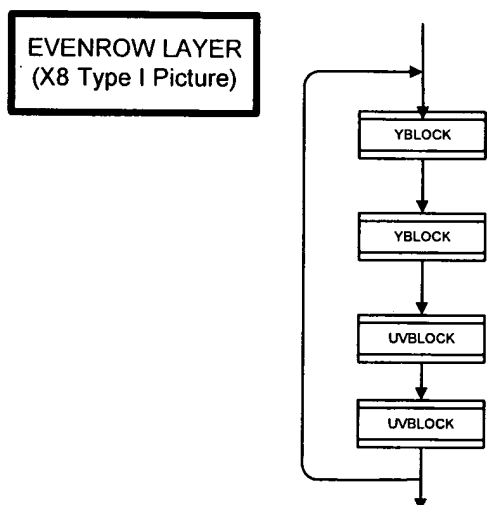
Figure 46: Syntax of even rows in an X8 intra frame

As shown in Figs. 25 and 26, the row layer is composed entirely of block layer information, i.e. there is no row layer stand-alone information.

The odd rows (beginning with the first row, numbered 1) carry only block layer luminance information (YBLOCK), whereas even rows carry block layer luminance and chrominance information (latter is labeled UVBLOCK). For the YUV transform data of a 32 x 32 image represented in Figure 44, the ordering of coded information is the following:

[Y11 Y12 Y13 Y14] [Y21 Y22 U11 V11 Y23 Y24 U12] [V12 Y31 Y32 Y33 Y34] [Y41 Y42 U21 V21 Y43 Y44 U22 V22]

The square brackets group items of the same row, shown here for clarity.

4.3.3 Block-level syntax

YBLOCK and UVBLOCK layers form the block-level information. These are represented respectively in the top and bottom of Figure 47. YBLOCK data is composed of variable length and optional DIFFORIENT, followed by a variable length DCVALUE and a possible succession of variable length ACVALUE symbols. UVBLOCK is similar to YBLOCK, except there is no DIFFORIENT symbol.

DIFFORIENT denotes the differential spatial orientation of the block, and is inserted whenever the previously transmitted blocks surrounding the current block satisfy a certain boundary variance condition. DCVALUE and ACVALUE are symbols jointly coding (*level, last*) and (*run, level, last*). *level* is the quantized DCT level of DC coefficient for DCVALUE and AC coefficient for ACVALUE. *run* is the run of zero valued coefficients between the previous non-zero coefficient and current. *last* is a binary symbol indicating the termination of the YBLOCK or UVBLOCK layer when it is set to 1.
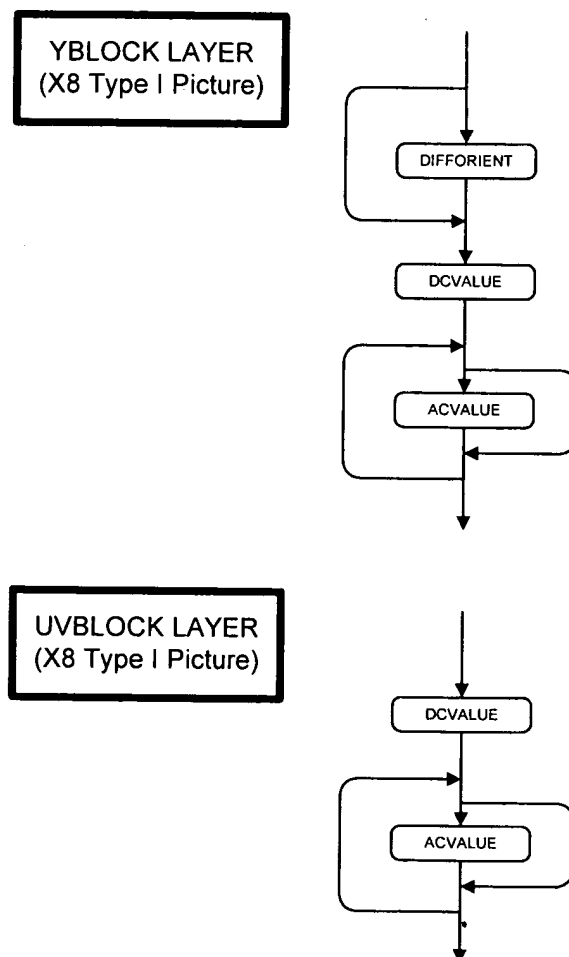
Figure 47: Syntax of blocks in an X8 intra frame

4.3.4 Spatial Prediction

Spatial prediction exploits redundancies over the spatial extent of the I frame. Here, prediction is performed on a block basis across the luminance channel. On the chrominance channels, coding takes place block by block, with the spatial prediction information being pulled from the corresponding top left luminance block. The specific operation of spatial prediction of a block is characterized by the *prediction mode*. In general, information regarding the prediction mode of a block is transmitted as part of the bitstream.

The encoding process takes place as follows: the current 8 × 8 block is predicted entirely from its causal neighbors given a certain prediction mode. The difference between the current block and its prediction is computed. The 8 ×

8 difference matrix is further coded using a linear transform (specifically, a modified version of the DCT). Decoding is the inverse process of encoding. The prediction mode and causal neighborhood are precisely known at the decoder. Using this information, the prediction of the current block is generated. The (lossy) difference matrix is regenerated from the bitstream, and is added to the prediction to generate the decoded block.
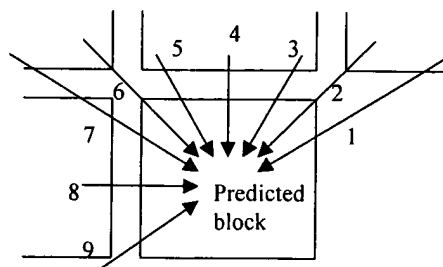
Figure 48: Prediction mode indices and prediction directions

4.3.4.1   Prediction Modes

The prediction modes are a set of rules that determine the "prediction" of the current block from its causal neighborhood. Each mode of prediction is associated with a distinct prediction, and one out of these modes is picked for encoding the current block. X8INTRA uses twelve prediction modes in all. These include horizontal, vertical and several diagonal extrapolations of the predicting edges (i.e. edge pixels of the causal neighbor blocks abutting the current block), smooth blends of horizontal and vertical predictions, and a bidirectional diffusion type operation called the *null* prediction. In addition, there is a prediction mode used when the causal blocks have negligible variation (the so called *flat condition*).

Figure 48 shows the prediction modes and corresponding prediciton directions. Mode 0 is the diffusion predictor, mode 4 is vertical and 8 is horizontal.

Figure 49 labels the neighbor pixels to aid definition and explanation of the processes.
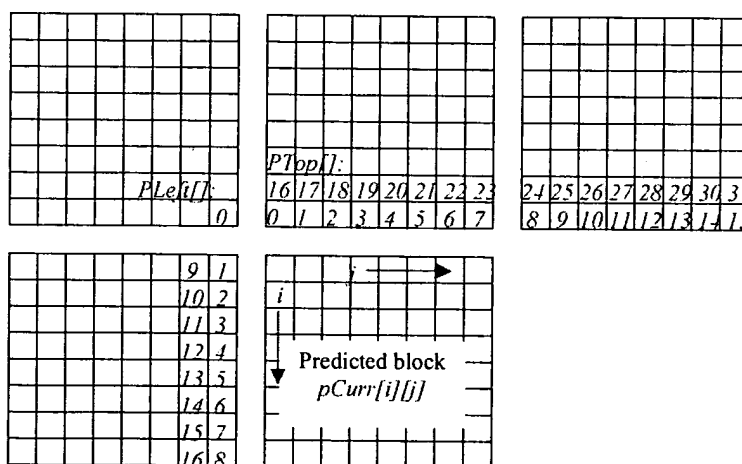
Figure 49: Predicting edge labels – indices i and j run from 0 through 7.

Setup

Prior to the prediction of a block, a set of operations are performed on the causal predicting edges generated from the reconstruction till the current block, i.e. the left and top blocks. This is carried out in the function setup, and proceeds as follows from item number 2.

Item number 1 predicts the orientation for the current block from the causal neighbors' orientation. Predicted orientation can only be null, horizontal or vertical.

1. If current block is the leftmost block in the top row, predicted orientation is null (0). Else if any other block in the top row, it is horizontal (8). If current block is any other block in the leftmost column, it is vertical (4). In other cases, it is as defined by the top left (TL), left (L) and top (T) block "meta-directions". Meta-directions are remapped from the twelve actual orientation directions according to Table 44 (top). Based on the meta-directions of L, T, TL and QP, Table 44 (bottom) lists the predicted orientation.

Table 44: Rules to determine predicted orientation

Orientation	Meta-direction
Horizontal (8)	H (8)
Vertical (4)	V (4)
All others	Null (0)

L	T	*predicted orientation* (PO)	notes
X	X	X	if identical, use same meta-direction
H	0	H	horizontal continuity from left
0	V	V	vertical continuity from top
H	V	H	horizontal continuity over-rides vertical

V	H	Code segment:
		
		`if (TL==L)  PO=T;`
		`else {`
		`   if (QP>12)  PO=T;`
		`   else {`
		`      if (TL==T)  PO=L;`
		`      else        PO=TL;`
		`   }`
		`}`	

otherwise		0	

2. pLeft[] and pTop[] arrays are populated using predicting edge data, according to Figure 49, except for top row and leftmost column blocks. If the block is at the top left corner, all neighbors pLeft[] and pTop[] are set to 128. If the block is on the top row (but not at the left extreme), pLeft[0] and pTop[] are set to pLeft[1]. If the block is in the first column (but not at the top extreme), all elements of pLeft[] are set to pTop[0]. The remaining neighboring elements are copied from the causal reconstructed neighbor blocks of the current color plane.

3. The range of the *immediate* neighbors is computed, i.e. maximum value minus minimum value of pLeft[1...8] and pTop[0...7] is found. If range is less than 3, the *flat prediction mode* is activated, and

a DC value iDcValue is computed as:

$$iDcValue = \left\lfloor \frac{6899 \cdot \left(\sum_{0 \le i \le 9} pTop[i] + \sum_{0 \le i \le 8} pLeft[i] + 9 \right)}{2^{17}} \right\rfloor \cdot$$

Else, if range is either less than *QP*, the *predicted orientation* of the block is reset to the null predictor. With this, setup concludes for color channels.

4. For luminance channels, if the range is smaller than 2*QP*, and the block is not on the top or left periphery of the image, horizontal and vertical predicted orientations are changed to blended horizontal and blended vertical orientations respectively. Also, if the range is smaller than 2*QP*, the orientation direction, DIFFORIENT, is *not transmitted* (or received). This ensures that bits are not wasted transmitting spatial orientation information if there is little information in the causal boundary to begin with.

Orientation information is not transmitted for the chrominance channels. Chrominance blocks use the meta-direction of the top-left block in the corresponding luminance macroblock.

$$c = \left\lfloor \frac{a+b+1}{2} \right\rfloor$$

Left causal block   Predicted block
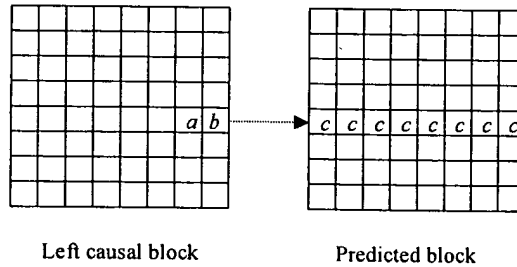
Figure 50: Horizontal prediction mode (Vertical prediction is the transpose)

Horizontal and Vertical Prediction Modes

The horizontal and vertical prediction modes are exact transposes of each other. For the horizontal prediction mode (mode index 8), the two rightmost columns in the reconstructed block to the left of the current are averaged, and this value copied across all columns of the prediction, as shown in the Figure 50.

Vertical prediction (mode index 4) is the transpose of horizontal prediction, i.e. the prediction values are copied down across all rows from the average of the bottom two rows of the top causal block.

Diagonal Prediction Modes

In all, there are seven diagonal prediction modes used in X8INTRA. These correspond to mode indices 1-3, 5-7 and 9, roughly corresponding to extrapolations in increments of approximately 22.5 degrees proceeding counter-clockwise. Figure 48 shows a diagram of prediction modes and corresponding extrapolation directions.

The following pseudo-code defines the predictors for diagonal prediction modes:

mode = 1:

```
pCurr[i][j] = pTop[(2*i + j + 2 > 15) ? 15 : 2*i + j + 2]
```

mode = 2:

```
pCurr[i][j] = pTop[i + j + 1]
```

mode = 3:

```
        pCurr[i][j] = pTop[floor((i + 1)/2) + j]
```

mode = 4:

```
        pCurr[i][j] = floor((pTop[j] + pTop[j + 16] + 1)/2)
```

mode = 5:

```
        if (j >= floor((i + 1)/2))
            pCurr[i][j] = pTop[j - floor((i + 1)/2)]
        else
            pCurr[i][j] = pLeft[i - 1 - 2*j]
```

mode = 6:

```
        if (j > i)
            pCurr[i][j] = pTop[j - i - 1]
        else
            pCurr[i][j] = pLeft[i - j]
```

mode = 7:

```
        if (j > 2*i + 1)
            pCurr[i][j] = floor((pTop[j-2*i-2] + pTop[j-2*i-1] + 1)/2)
        else
            pCurr[i][j] = pLeft[floor(j/2) - i]
```

mode = 8:

```
        pCurr[i][j] = floor((pLeft[i + 1] + pLeft[i + 9] + 1)/2)
```

mode = 9:

```
        if (i + j < 6)
            pCurr[i][j] = pLeft[i + j + 2]
        else
            pCurr[i][j] = pLeft[8]
```

Blended Prediction Modes

Prediction modes (indices 10 and 11) try to blend features from both the top and left blocks while predicting the current block. The blend is a linear combination of corresponding horizontal and vertical prediction edge pixels, as defined in the following pseudo code.

mode = 10: blended horizontal continuity

```
        pCurr[i][j] = (pLeft[i + 1]*(8 - j) + pTop[j]*j + 4) >> 3
```

mode = 11: blended vertical continuity

```
        pCurr[i][j] = (pLeft[i + 1]*i + pTop[j]*(8 - i) + 4) >> 3
```

Null Prediction Mode

The null prediction mode, with mode index 0, attempts to extrapolate the current block from its causal neighbors independent of direction. The idea here is to predict pixel pCurr[i][j] as a linear combination of pTop and pLeft elements, with weights being proportional to a negative exponent of the distance. The current predicted pixel is modeled as a linear sum of one element from each of two cumulant arrays.

The first stage of building the cumulant arrays is to set up an array of weights roughly corresponding to lowpass filtered left and top predicting edge pixels. These arrays are labeled pLeftSum and pTopSum respectively. This operation is represented in C below.

```
        for (int i = 0; i < 8; i++) {
            int ll = (i < 4) ? 8 : 4 + i;
            sl[0] = sl[1] = st[0] = st[1] = 0;

            for (int k = 0; k < ll; k++) {
                int diff = abs(k - i);
                int wt = 4 - (diff >> 1);
                st[diff & 1] += ((int) pTop[k]) << wt;
```

```
if (k < 8)
        sl[diff & 1] += ((int) pLeft[k + 1]) << wt;

k++;
diff = abs(k - i);
wt = 4 - (diff >> 1);
st[diff & 1] += ((int) pTop[k]) << wt;
if (k < 8)
        sl[diff & 1] += ((int) pLeft[k + 1]) << wt;
}


pTopSum[i]  = st[0] + ((st[1] * 181 + 128) >> 8);
pLeftSum[i] = sl[0] + ((sl[1] * 181 + 128) >> 8);
}
```

Once the arrays pLeftSum and pTopSum are set up, the predicted block is computed by summing the appropriate element from each array, using the following rule.

```
pCurr[i][j] =
(pTopSum[j] * pWtsT[i][j] + pLeftSum[i] * pWtsL[i][j] + 32768) >> 16;
```

where

```
pWtsT =
    640   669   708   748   792   760   808   772
    480   537   598   661   719   707   768   745
    354   416   488   564   634   642   716   706
    257   316   388   469   543   571   655   660
    198   250   317   395   469   507   597   616
    161   206   266   340   411   455   548   576
    122   159   211   276   341   389   483   520
    110   144   193   254   317   366   458   499

pWtsL =
    640   480   354   257   198   143   101    72
    669   537   416   316   250   185   134    97
    708   598   488   388   317   241   179   132
    748   661   564   469   395   311   238   180
    792   719   634   543   469   380   299   231
    855   788   710   623   548   455   366   288
    972   914   842   758   682   584   483   390
   1172  1107  1028   932   846   731   611   499
```

4.3.4.2   Orientation Transmission

The orientation of a block which determines its spatial prediction from its neighbors is coded and transmitted only for luminance blocks, and when it is of relevance (as determined by setup). If so, only the difference between the computed orientation and its predictor is sent. This difference is characterized by three arrays refered to the null, horizontal and vertical *orderings*. These arrays are reproduced below.

```
int orderArray[]  ={0,8,4,  10,11,  2,6,  9,1,3,  5,7};
int orderArrayH[] ={8,0,4,  10,11,  1,7,  2,6,9,  3,5};
int orderArrayV[] ={4,0,8,  11,10,  3,5,  2,6,9,  1,7};
```

The direction of the predicted orientation determines which of the orderings is chosen. For example, if the predicted orientation is horizontal, orderArrayH is chosen. At the decoder, if the differential orientation index 7 is received for a block whose predicted orientation is horizontal, orderArrayH[7] gives its actual orientation to be 2 (diagonal from the top-right). Orderings are designed for coding efficiency. The differential orientation is coded as the DIFFORIENT symbol in the syntax diagram, using a Huffman code as explained in Section 4.3.8.2.

4.3.5 Skewed IDCT

The skewed IDCT (SDCT) exploits non-stationarity across pixels in a block, in the encoding of the block. The skewed IDCT is a modification of the IDCT. Three types of skew, relating to the horizontal, vertical and null prediction modes are used here. The horizontal and vertical SDCTs are skewed in one dimension only, whereas the null SDCT is skewed in 2D. Also, the horizontal and vertical skews are transposes.

See section 4.11 regarding IDCT conformance.

4.3.5.1 Implementation

The horizontal SDCT is used for the horizontal predictor as well as predictors that are "largely" horizontal (i.e. prediction orientation of $\pm\pi/8$ to the horizontal axis). These are prediction modes 8, 9 and 10. Likewise, the vertical SDCT is used for the vertical and near-vertical predictors, which are modes 3, 4 and 11. The null SDCT is used for the null (0), 5, 6 and 7 prediction modes. Prediction modes 1 and 2 use the unmodified IDCT. The same rules apply to the chroma channels as well.

However, blocks on the top row and left column use the unmodified IDCT. Also, blocks whose quantized DC residual value is –1, 0 or 1 also use the unmodified IDCT.

The SDCT can be defined in a straightforward manner as follows, where the subscript * of B (0,0) is H, V, or 0, corresponding to the horizontal, vertical and null predictors respectively.

$$SDCT(T) = T(0,0)\, B_*(0,0) + \sum_{\substack{i,j=0\cdots7 \\ i+j>0}} T(i,j)B(i,j)$$
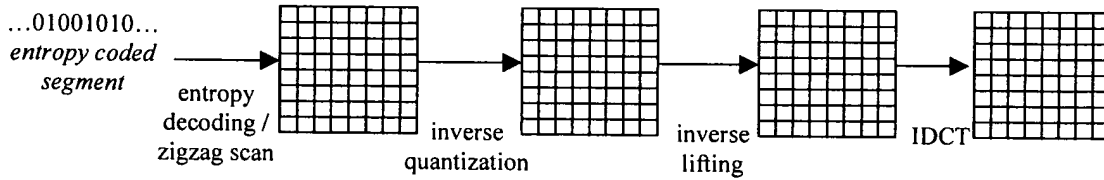
Figure 51: Decoding steps – inverse quantization, SDCT = { inverse lifting and inverse DCT }

4.3.5.2 Inverse Lifting

Inverse lifting is the method used in the X_INTRA_8 decoder to implement the SDCT. As shown in Figure 51, inverse lifting is a processing step inserted between inverse quantization and the IDCT. This stage modifies the inverse quantized transform coefficients to weight the DC response at pixels distant from the predicting edges.

The null, horizontal and vertical SDCTs are implemented with null, horizontal and vertical inverse lifting modes. The unmodified IDCT does not use lifting.

The horizontal and vertical lifting modes are transposes of each other. Let the input inverse quantized transform coefficient matrix for the current block be denoted by *pBlock[i][j]*, *i* and *j* vary from 0 through 7. The horizontal lifting mode modifies four coefficients of the block according to:

$$pBlock[0][1] = pBlock[0][1] - \left\lfloor \frac{6269 \cdot pBlock[0][0] + 32768}{65536} \right\rfloor$$

$$pBlock[0][3] = pBlock[0][3] - \left\lfloor \frac{708 \cdot pBlock[0][0] + 32768}{65536} \right\rfloor$$

$$pBlock[0][5] = pBlock[0][5] - \left\lfloor \frac{172 \cdot pBlock[0][0] + 32768}{65536} \right\rfloor$$

$$pBlock[0][7] = pBlock[0][7] - \left\lfloor \frac{73 \cdot pBlock[0][0] + 32768}{65536} \right\rfloor$$

The null lifting mode uses the following rule

$$pBlock[i][j] = pBlock[i][j] - \mathrm{sgn}(pBwt[i][j]) \cdot \left\lfloor \frac{|pBwt[i][j]| \cdot pBlock[0][0] + 32768}{65536} \right\rfloor$$

where *pBwt* is the 8×8 array of weights

$$pBwt = \begin{pmatrix} 0 & 3811 & 487 & 506 & 135 & 173 & 61 & 42 \\ 3811 & -1084 & -135 & -135 & -42 & -61 & 0 & 0 \\ 487 & -135 & 0 & 0 & 0 & 0 & 0 & 0 \\ 506 & -135 & 0 & 0 & 0 & 0 & 0 & 0 \\ 135 & -42 & 0 & 0 & -42 & 0 & 0 & 0 \\ 173 & -61 & 0 & 0 & 0 & 0 & 0 & 0 \\ 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 42 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

4.3.5.3   Flat condition

The *flat condition* is a particular situation where the skewed transform presents a liability rather than an advantage. In such situations, the non-skewed IDCT is used. The *flat condition* is indicated by a coincidence of (i) range less than 3 in `setup`, (ii) quantized DC coefficient being −1, 0 or 1, and (iii) no nonzero AC coefficients.

Adjustment for the *flat condition* proceeds by setting all pixels in the block to a common DC value. This DC value is determined as shown in the code below, where `iDC` is the quantized DC coefficient, and the predicting edge DC value `iDcValue` is determined during `setup`. After adjusting for `iDC`, the DC value is stored back in `iDcValue`. The function `clamp()` returns its integer argument clamped between 0 and 255.

```
iDC += x8QuantizeDC(iDcValue << 3);
iDcValue = clamp((x8DequantizeDC(iDC) + 4) >> 3);
```

Quantization and dequantization of DC coefficients (`x8QuantizeDC` and `x8DequantizeDC`) is defined in Section 4.3.6.1. The *flat condition* triggers the flat prediction mode in which all pixels in the predicted block are set to `iDcValue`.

4.3.6 Transform Coefficient Encoding

This section deals with the encoding of quantized differential transform coefficients, DCVALUE and ACVALUE. This step occurs subsequent to spatial prediction and skewed DCT on the encoder side (i.e. prior to SDCT and spatial prediction on the decoder side). For this stage, the input is an 8 × 8 block of quantized coefficients. This section explains the decisions and criteria related to coding information pertinent to this block of data.

4.3.6.1   Dequantization

The quantization parameter (QP) is an index that determines quantization step sizes of the various quantities undergoing quantization and dequantization. The permissible quantization parameter (QP) values for X8INTRA are 6 through 31.

Dequantization and Quantization of DC terms

`stepSize` and `stepSizeC` are quantization step sizes for DC terms in the luminance and chrominance channels respectively. They are determined from QP using

```
stepSize = QP;
stepSizeC = ((9 * QP + 3) >> 3);
```

The DC transform coefficient of a block is quantized by a flat quantizer. The reconstruction rule `x8DequantizeDC()` to generate *y* given the quantized coefficient *x* is $y = x * $ `stepSize` (or `stepSizeC` for chrominance).

Although quantization is typically an encoder-only issue, handling of the *flat condition* requires that it be defined at the decoder as well. Quantization of DC coefficients proceeds by first computing an integer inverse *QP*

$$iQP = \left\lfloor \frac{65536 + \left\lfloor \frac{step}{2} \right\rfloor}{step} \right\rfloor .$$

step in the above equation is `stepSize` or `stepSizeC`, based on the current colorplane. The quantized value

y=`x8QuantizeDC`(*x*) corresponding to raw integer DC coefficient *x* is $\left\lfloor \frac{x \cdot iQP + 32768}{65536} \right\rfloor$.

Dequantization of AC terms

When the quantized input AC transform coefficient is *x*, the dequantized reconstruction *y* is given by

$$y = \left\lfloor \frac{(x \cdot 2 \cdot QP + \text{sgn}(x) \cdot QP1)R}{256} \right\rfloor$$

where *QP* is the quantization parameter. *QP*1 is equal to *QP* for odd values, and to *QP*-1 for even values of *QP*.

R is a reconstruction value that is either a constant for all transform coefficients or a position-dependent value. The former case is the default mode of operation, while the latter case is termed *non-flat dequantization*. This is signalled by NONFLAT, whose values of 0 and 1 correspond respectively to the two cases. In the default mode (NONFLAT = 0), *R* is 256. In this mode, the division and round-down step may be eliminated. For *non-flat* dequantization (NONFLAT = 1), the value of *R* is determined from the array `gaReconstructionLevels[]` reproduced below. The element of the array to be used is the index of the transform coefficient in the zigzag scan, counting the DC coefficient as well.

```
int gaReconstructionLevels[] = {
    256,256,256,256,256,256,259,262,  265,269,272,275,278,282,285,288,
    292,295,299,303,306,310,314,317,  321,325,329,333,337,341,345,349,
    353,358,362,366,371,375,379,384,  389,393,398,403,408,413,417,422,
    428,433,438,443,448,454,459,465,  470,476,482,488,493,499,505,511};
```

4.3.6.2 Scan order

One of three scan orders are used to encode each quantized transform block. The three scan orders are respectively the null, horizontal and vertical scan orders. The meta-direction of the block orientation (or predicted orientation, for blocks which do not have an associated differential orientation) determines the scan order, which are shown below.

Null scan order:

```
int grgiZigzagInv_NEW[] = {
     0,  8,  1,  2,  9,16,24,17,  10,  3,  4,11,18,25,32,40,
    48,56,41,33,26,19,12,  5,    6,13,20,27,34,49,57,58,
    50,42,35,28,21,14,  7,15,   22,29,36,43,51,59,60,52,
    44,37,30,23,31,38,45,53,    61,62,54,46,39,47,55,63};
```

Horizontal scan order:

```
int grgiHorizontalZigzagInv_NEW[] = {
     0,  1,  8,  2,  3,  9,16,24,  17,10,  4,  5,11,18,25,32,
    40,48,33,26,19,12,  6,  7,    13,20,27,34,41,56,49,57,
    42,35,28,21,14,15,22,29,     36,43,50,58,51,44,37,30,
    23,31,38,45,52,59,60,53,     46,39,47,54,61,62,55,63};
```

Vertical scan order:

```
int grgiVerticalZigzagInv_NEW[] = {
     0,  8,16,  1,24,32,40,  9,   2,  3,10,17,25,48,56,41,
    33,26,18,11,  4,  5,12,19,    27,34,49,57,50,42,35,28,
    20,13,  6,  7,14,21,29,36,    43,51,58,59,52,44,37,30,
    22,15,23,31,38,45,60,53,     46,39,47,54,61,62,55,63};
```

The above arrays are inverse scan orderings. The array index corresponds to zigzag scan order, and array entry at that index is the transform coefficient within the block. In this case, the transform coefficients are reordered, row-

by-row, into a linear array of size 64. For example, the 16th element of the horizontal zigzag scan (starting with zero) points to the 40th linearized transform coefficient, which is the first element of the fifth row (indexed from 1).

Note that the horizontal and vertical scan orders are *not* transposes.

4.3.6.3   Significant coefficient estimation

While encoding transform coefficients within a block, the first N number of AC coefficients is encoded using a certain entropy table and the remainder uses a different entropy table. The number N is determined from a prediction of the number of significant coefficients in the block, from its causal neighbors. This prediction is available at the decoder and thus forms a valid context for encoding. This context is used for both DC and AC coefficients.

N is the minimum of the number of nonzero AC coefficients in the three blocks to the left, top and top-left of the current block. For blocks on the top row, N is the number of nonzero AC coefficients in the block to the left. Similarly, for blocks on the leftmost column it is the number of nonzero AC coefficients in the block at the top. For the top left corner block, N is 16.

The luminance block DCVALUE is coded using one of two tables depending on whether N is zero or not. When N is zero, the LH_INTRAZ *type* is used, whereas when N is positive, LH_INTRANZ type is used. The *type* is a context that determines which coding table is used by the symbol, and is explained in Section 4.3.8.3. Chrominance block DC coefficients use the LH_INTRAC0 type, regardless of N.

Likewise, *types* are also defined for quantized AC coefficients. Chrominance ACVALUE symbols use the LH_INTER type. For ACVALUE symbols of luminance blocks whose differential orientation index DIFFORIENT is greater than 4, the LH_INTER0 type is used. For other ACVALUE symbols in luminance blocks, LH_INTRAY0 and LH_INTRAY are used, respectively when the current symbol is less than N or not. The current symbol is the running count of nonzero ACVALUE symbols in the current block, starting with 1. Further details are presented in Section 4.3.8.4.

4.3.7 Loop Filtering

This process is performed if and only if LOOPFILTER is set.

Upon decoding an 8×8 block in either luminance or chrominance planes, the left and top edge of the block is subjected to a filtering process to reduce "blockiness". It is necessary to perform deblocking immediately after decoding the block in order for spatial prediction to work correctly.

In the following figure, block boundaries are marked by bold lines. Horizontal and vertical deblocking filters are transposes – only the horizontal deblocking filter is explained here. The horizontal deblocking filter operates on a left-right pair of blocks, and the vertical on a top-bottom pair.
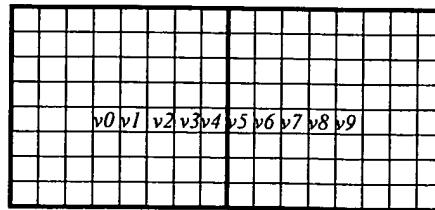
Figure 52: Horizontal deblocking filter explained

Two deblocking modes are used – one mode applies a *short* filter to one pixel on either side of the block edge whereas the other mode applies a *long* filter to two pixels on either side. For each of the eight horizontal edge pixel-pairs labeled as *v4-v5* in the above figure, an edge strength E is computed

$$E = \phi(v0-v1)+\phi(v1-v2)+\phi(v2-v3)+\phi(v3-v4)+\phi(v4-v5)+\phi(v5-v6)+\phi(v6-v7)+\phi(v7-v8)+\phi(v8-v9)$$

$$\phi(x) = \begin{cases} 1 & \text{if } |x| \le T \\ 0 & \text{otherwise} \end{cases}$$

$$T = \left\lfloor \frac{QP+10}{8} \right\rfloor$$

If E is less than 6, the *short* filter is chosen. The short filter is defined in Section 4.4.

If E is greater than or equal to 6, then the range of the values *v1* through *v8* is computed. Range is defined as the maximum minus the minimum value of these eight variables. If the range is greater than or equal to $2QP$, the *short* filter is used. Else, the long filter is applied. In the long filter, *v3* through *v6* are modified as follows

$$v3' = \left\lfloor \frac{4v2+3v3+v7+4}{8} \right\rfloor$$

$$v4' = \left\lfloor \frac{3v2+3v4+2v7+4}{8} \right\rfloor$$

$$v5' = \left\lfloor \frac{3v7+3v5+2v2+4}{8} \right\rfloor$$

$$v6' = \left\lfloor \frac{4v7+3v6+v2+4}{8} \right\rfloor$$

No vertical deblocking is performed for the top row of blocks, and no horizontal deblocking for the left column. Horizontal deblocking is followed by vertical deblocking for the block.

The following segment of C code illustrates the enabling of loop filtering. Arguments `blockIndexX` and `blockIndexY` are horizontal and vertical indices of the current block. Boolean `bResidual` is true when *flat condition* is true, or when there is at least one non-zero DCT term (AC or DC). `iOrient` is the absolute orientation direction.

```
deblock(int blockIndexX, int blockIndexY, bool bResidual, int iOrient)
{
    if ((blockIndexY > 0) && (bResidual || (iOrient != 0 && iOrient != 4)))
        horizontalDeblockFilter();
    if ((blockIndexX > 0) && (bResidual || (iOrient != 0 && iOrient != 8)))
        verticalDeblockFilter();
}
```

4.3.8 Entropy Coding

Source symbols are transmitted by means of Huffman codes. Entropy coded source symbols in X8INTRA are of DIFFORIENT, ACVALUE and DCVALUE categories.

Huffman tables for X8INTRA symbols are provided in Section 5.1.

4.3.8.1 Table switching

Within each category of symbols, there may be a further sub-division refered to as *type*, explained later. Multiple Huffman tables are used for each *type* of symbols (or category if the *type* subdivsion does not exist). With each *type* or subdivision is associated a certain set of Huffman tables. The cardinality of this set is 2^k. The category and *type* of a symbol can be wholly inferred from the causal information available at the decoder.

In any given I frame, all symbols of a certain *type* (or category) are encoded using one Huffman table chosen out of the candidate set. The index of this table within the set is indicated by a fixed length code which is the index represented in k bits. For example, if the third (i.e. index = 2) out of eight tables is chosen for a certain *type*, the first symbol of this *type* is preceded by 010.

Huffman code tables used in X8INTRA are represented as doubles of codewords and code lengths. Codewords are shown in hexadecimal.

QP-based *types*

The *type* of a symbol includes a dependence on QP. "*low-rate*" refers to QP > 12 and "*high-rate*" to QP <= 12 conditions.

4.3.8.2 Orientation Coding

DIFFORIENT, when relevant, is encoded using a Huffman code. Two *types* are defined, corresponding to *low-rate* and *high-rate* categories. These are encoded using a set of two (Table 60) and four (Table 61) Huffman tables respectively.

4.3.8.3 DCVALUE (*level-last*) Coding

Block transform coding reduces most transform coefficients to zero after quantization. The quantized transform coefficients are coded using run length coding. In this method, the first coefficient (absolute value *level*) is coded jointly with a binary symbol *last* that signals whether there are any subsequent coefficients (*last* = FALSE) or not (*last* = TRUE).

The permissible range of *level* in X8INTRA DCVALUE coefficients is –256 through 256, including 0.

index-fine representation

The joint symbol *level-last* is translated into a bin *index* and a *fine* address within the bin. The size of each bin, i.e. the number of joint *level-last* symbols in each bin labeled by *index* is known at the decoder, and is 2^k, for some integer k. The *fine* address is k bits long, uniquely and efficiently specifying the symbol. Table 45 shows the mapping between the absolute *level* and bin *index*. The latter spans the integers 0 through 33, with the first 17 indices corresponding to *last* = FALSE.

level = 0 is a permissible value, and the sign of the coefficient is coded in one bit if *level* is nonzero. The size of the *fine* address includes the bit needed to code sign of *level* which is coded in the LSB of *fine*. The upper bits of *fine* locate the absolute value within the bin, in the natural ordering. For example, *level-last* = (-28, TRUE) translates to (10, 7) in *index-fine* space. *index* is coded using a Huffman code.

Table 45: *level-last* to *index-fine* mapping

last	FALSE	TRUE	
|*level*|	*index*	*index*	*fine* size
0	0	17	0
1	1	18	1
2	2	19	1
3	3	20	1
4	4	21	1
5-6	5	22	2
7-8	6	23	2
9-12	7	24	3
13-16	8	25	3
17-24	9	26	4
25-32	10	27	4
33-48	11	28	5
49-64	12	29	5
65-96	13	30	6
97-128	14	31	6
129-192	15	32	7
193-256	16	33	7

Huffman coding

Six types are defined in all for the DCVALUE category, three each for *low-rate* and *high-rate* scenarios. For each of the latter, symbols are divided into LH_INTRAZ, LH_INTRANZ and LH_INTRAC0 *types* according to rules in Section 4.3.6.3.

Huffman code tables for each *type* are drawn from a set of eight tables. The set corresponding to *low-rate* is shown in Table 62, and *high-rate* in Table 63.

4.3.8.4 ACVALUE / *run-level-last* Coding

The permissible range of level in X8INTRA ACVALUE coefficients is –128 through 128, and excludes 0.

The first coded symbol (DCVALUE) in the transform block indicates whether there are subsequent ACVALUE symbols. If there are, these symbols are run-length coded as a combination of three values viz. *run*, *level* and *last*. The *run* corresponds to the number of zero valued transform coefficients separating the current coefficient from the previously coded coefficient. *Level* is the magnitude of the current (nonzero) coefficient and *last* is a boolean variable denoting whether the current coefficient is the last in the current block. The *run-level-last* triple is referred to as RLL in the remainder of this Section.

index-fine representation

The RLL space is mapped into an alternate space called *index-fine*. *index* is an address that partitions the RLL space into several bins; each bin contains 2^k symbols. Within each bin, symbols have a nearly identical probability of occurrence. Hence, given the index with 2^k symbols in the bin, k bits uniquely identify the symbol.

X8INTRA partitions the RLL alphabet into 77 bins. The first 46 bins (labeled 0 through 45) contain only one symbol triple, whereas bins 46 through 76 contain 2^k symbols, where $k > 0$. The multi-symbols bins are labeled, in sequence, as

```
Index 46 through 58
L0a, L0b, L0c, L0d, L0e, L0f, L0z, L0y, L0x, L0w, L0v, L0u, L0t
Index 59 through 72
L1a, L1b, L1c, L1d, L1e, L1f, L1g, L1h, L1i, L1z, L1y, L1x, L1w, L1v
Index 73 and 74
Q0, Q1
Index 75 and 76
X0, X1.
```

Labels of the form L0* correspond to symbol triples whose *last* = FALSE (likewise L1* correspond to symbol triples whose *last* = TRUE). Bins with labels L* are linear, i.e. all symbols in a given bin have either the same *run* or the same *level*. Bins Q0 and Q1 each have 32 symbols. Table 46 and Table 47 illustrate the correspondence between *run-level-last* and *index* for *last*=0 and *last*=1 respectively. Table 48 shows the *fine* labels within Q0 and Q1.

X0 and X1 are escape symbols – X0 is a "short" escape used when *level* is less than 17, whereas X1 covers the whole RLL space. The *fine* codewords are of length 11 and 14 respectively, and are fixed length representations of *run-level-last* shown in Table 49.

Within any linear label of size 2^k symbols, the RLL symbol triple is specified by a k bit fixed length codeword. The symbols are in the natural ordering within the bin, i.e. numerically smaller codewords correspond to smaller *run* or *level*. For example, L0a is a linear bin with *last* = FALSE and *level* = 1. The eight symbols within this bin correspond to *run* = 16, 17 ... 23. A 3 bit fixed length codeword is used to specify the symbol within L0a. If this codeword is 5, the specified symbol in RLL representation is 21-1-FALSE, or 21-1-0 replacing the boolean with a binary variable.

For the most commonly occuring RLLs, *index* is between 0 and 45, hence *fine* is zero length. A Huffman code is used to encode *index*.

Table 46: RLL to index-fine mapping for *last*=FALSE

run / level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16-23	24-31	32-63
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	L0a	L0b	L0e
2	16	17	18	19	L0c				L0d								L0f		
3	20	21																	
4	22																		
5		L0			Q0														
6	L0	u																	
7	z																		
8																			
9																			
10	L0																		
11	y	L0																	
12		t																	
13																			
14	L0																		
15	x																		
16																			
17-24	L0 w																		
25-32	L0 v																		

Table 47: RLL to index-fine mapping for *last*=TRUE

run / level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16-19	20-23	24-27	28-31	32-47	48-63
1	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	L1a	L1b	L1c	L1d	L1e	L1f
2	39	40	41	42	L1g				L1h								L1i					
3	43	44																				
4	45																					
5		L1			Q1																	
6	L1	w																				
7	z																					
8																						
9																						
10																						
11		L1																				
12	L1	v																				
13	y																					
14																						
15																						
16																						
17-32	L1 x																					

Table 48: *fine* indices within Q0 and Q1

run		2	3	4	5	6	7	8	9	10
level	3	0	8	9	10	11	12	13	14	15
	4	1	16	17	24	25	26	27		
	5	2	18	19	28	29	30	31		
	6	3	20	21						
	7	4	22	23						
	8	5								
	9	6								
	10	7								

Table 49: *fine* codewords for escape symbols X0 and X1

X0	4 bits	6 bits	1 bit
X1	7 bits	6 bits	1 bit
	level	*run*	*last*

ACVALUE sign coding

The sign of the coefficient of ACVALUE is coded as a single bit after the *index-fine* code, with the bit being set for negative values of coefficient.

Huffman coding

In each *low-rate* and *high-rate* situations, one of four *types* are defined for ACVALUEs, labeled LH_INTER0, LH_INTER, LH_INTRAY0 and LH_INTRAY. The LH_INTRA* Huffman code type is typically associated with a higher entropy than the LH_INTER* code type. The LH_INTER* type codes are drawn from a set of eight tables S_INTER, and the LH_INTRA* type codes are drawn from another set of eight tables S_INTRA.

S_INTER Huffman table sets for *low-rate* and *high-rate* are given in Table 64 and Table 65 respectively. S_INTRA are likewise given in Table 66 and Table 67 respectively.

4.4 Progressive P Frame Decoding

Figure 58 shows the steps required to decode and reconstruct blocks in P frames. The following sections describe the process for decoding P pictures.

4.4.1 Out-of-bounds Reference Pixels

The previously decoded frame is used as the reference for motion-compensated predictive coding of the current P frame. The motion vectors used to locate the predicted blocks in the reference frame my include pixel locations that are outside the boundary of the reference frame. In these cases, the out-of-bounds pixel values are the replicated values of the edge pixel. Figure 53 illustrates pixel replication for the upper-left corner of the frame.

```
                                                    .     .
                                              .... 28 28 28   28 32 36 34 33 ...
                                              .... 28 28 28   28 32 36 34 33 ...
                                              .... 28 28 28   28 32 36 34 33 ...
      .... 28 28 28 | 28 32 36 34 33 ...       .... 28 28 28 | 28 32 36 34 33 ...
      .... 32 32 32 | 32 34 33 36 39 ...
```

            Horizontal pixel padding   Vertical pixel padding
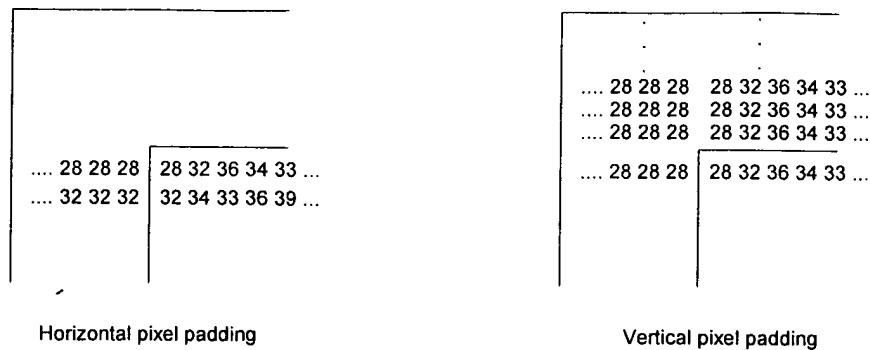
Figure 53: Horizontal and vertical pixel replication for out-of-bounds reference

4.4.2 P Picture Types

P pictures can be one of 2 types: 1-MV, Mixed-MV. The following sections describe each P picture type.

1-MV P Picture

In 1-MV P pictures, a single motion vector is used to indicate the displacement of the predicted blocks for all 6 blocks in the macroblock. The 1-MV mode is signaled by the MVMODE and MVMODE2 picture layer fields as described in section 4.4.3.3. 1-MV mode is used in interlace decoding (INTERCLEN = 1 in sequence layer) if the picture is coded in frame mode – indicated by picture layer field INTRLCF = 0.

Mixed-MV P Picture

In Mixed-MV P pictures, each macroblock can be encoded as a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luminance macroblocks has a motion vector associated with it. The 1-MV or 4-MV mode for each macroblock is indicated by the MVTYPEMB bitplane field in the picture layer as described in section 4.4.3.3. The Mixed-MV mode is signaled by the MVMODE and MVMODE2 picture layer fields as described in section 4.4.3.3.

4.4.3 P Picture Layer Decode

Figure 10 shows the elements that make up the progressive P picture layer header. Some of the elements are self-explanatory. The following sections provide extra detail for some of the elements.

4.4.3.1 Picture-level Quantizer Scale

The frame level quantizer scale PQUANT is decoded from the 5-bit picture layer field PQINDEX as described in section 3.2.1.7. PQUANT specifies the frame level quantizer scale (a value between 1 and 31) for the macroblocks in the current picture. When the seq. header DQUANT = 0, then PQUANT is used as the quantization step size for every macroblock in the current picture. When DQUANT != 0, then PQUANT is used as described in section 3.2.1.27. The PQINDEX field also specifies whether the 3-QP or 5-QP deadzone quantizer is used for all macroblocks in the frame.

4.4.3.2 Picture Resolution Index

The RESPIC field in P pictures specifies the scaling factor of the decoded P picture relative to a full resolution frame. The decoded picture may be full resolution or half the original resolution in either the horizontal or vertical dimensions or half resolution in both dimensions. Table 8 shows how the scaling factor is encoded in the RESPIC field.

If the picture resolution for the current P frame is different than the resolution of the previous frame then the reference frame must be resampled to match the new frame resolution. The process described in section 4.1.1.6 must be used to resample the reference frame.

4.4.3.3 Picture Layer Motion Compensation and Intensity Compensation Decoding

The P picture layer contains syntax elements that control the motion compensation mode and intensity compensation for the frame. The MVMODE field is a variable sized value that signals either: 1) one of four motion vector modes for the frame or 2) that intensity compensation is used in the frame. If intensity compensation is

signaled then the MVMODE2, LUMSCALE and LUMSHIFT fields follow in the picture layer. In this case, MVMODE2 signals the motion vector mode and LUMSCALE and LUMSHIFT are 6-bit values which specify parameters used in the intensity compensation process. Refer to section 4.4.7 for a description of intensity compensation decode.

Table 11 and Table 12 show the codetables used to decode the MVMODE and MVMODE2 fields. Table 11 is used if PQUANT is greater than 12 and Table 12 is used if PQUANT is less than or equal to 12. Either MVODE or MVMODE2 will signal one of four motion vector modes. If the motion vector mode is mixed MV mode then the MVTYPEMB field is present in the picture layer. MVTYPEMB is a bitplane coded field that indicates the 1-MV/4-MV motion vector status for each macroblock in the picture. The decoded bitplane represents the motion vector status for each macroblock as a field of 1-bit values in raster scan order from upper left to lower right. Refer to section 4.10 for a description of the bitplane coding. A value of 0 indicates that the macroblock is coded in 1-MV mode. A value of 1 indicates that the macroblock is coded in 4-MV mode. Refer to section 4.4.4.2 for a description of the motion vector decoding process.

4.4.3.4 Skipped Macroblock Decoding

The P picture layer contains the SKIPMB field which is a bitplane coded field that indicates the skipped/not-skipped status of each macroblock in the picture. The decoded bitplane represents the skipped/not-skipped status for each macroblock as a field of 1-bit values in raster scan order from upper left to lower right. Refer to section 4.10 for a description of the bitplane coding. A value of 0 indicates that the macroblock is not skipped. A value of 1 indicates that the macroblock is coded as skipped. A skipped status for a macroblock means that the macroblock may contain the HYBRIDPRED field. Refer to section 4.4.4.2 for a description of how the HYBRIDPRED field is used in the decoding process.

4.4.3.5 Motion Vector Huffman Table

MVTAB is 2-bit field in the picture layer that indicates the Huffman table used to decode the motion vector differentials for the macroblocks in the picture. The Huffman tables are encoded as shown in Table 50. Section 5.7 contains the Motion Vector Differential Huffman tables. Refer to section 4.4.4.2 for a description of the motion vector decode process.

Table 50: Motion vector Huffman table

MVTAB FLC	Huffman table
00	Motion Vector Table 0
01	Motion Vector Table 1
10	Motion Vector Table 2
11	Motion Vector Table 3

4.4.3.6 Coded Block Pattern Huffman Table

CBPTAB is 2-bit field in the picture layer that indicates the Huffman table used to decode the coded block pattern (CBPCY) for the macroblocks in the picture. The Huffman tables are encoded as shown in Table 51. Section 5.3 contains the CBP Huffman tables. See section 4.4.4.2 for a description of how CBPCY is used.

Table 51: CBP Huffman table

CBPTAB FLC	Huffman table
00	CBP Table 0
01	CBP Table 1
10	CBP Table 2

11	CBP Table 3

4.4.3.7 Macroblock-level Quantizer Mode Flag

See section 3.2.2.6.

4.4.3.8 Macroblock-level Transform Type Flag

TTMBF is a one-bit field that signals whether transform type coding is enabled at the frame or macroblock level. If TTMBF = 0 then the same transform type is used for all blocks in the frame. In this case, the transform type is signaled in the TTFRM field that follows. If TTMBF = 1 then the transform type can vary throughout the frame and is signaled at the macroblock or block levels.

4.4.3.9 Frame-level Transform Type

TTFRM is a variable-length field that is present in the picture layer if TTMBF = 1. TTFRM is decoded using Table 19 and signals the DCT transform type used to transform the 8x8 pixel error signal in predicted blocks. The 8x8 error blocks can be transformed using an 8x8 DCT, two 8x4 DCTs, two 4x8 DCTs or four 4x4 DCTs.

4.4.3.10 Macroblock-level DCT AC Coding Set Flag

DCTACMBF is a one-bit field that signals whether the coding set used to decode the DCT AC coefficients is signaled at the frame level or macroblock level. If DCTACMBF = 1 then the coding set used to decode all the AC coefficients in that macroblock is signaled at the macroblock level. If DCTACMBF = 0 then the coding set is used for all the blocks in the frame. In this latter case, the coding set is signaled in the DCTACFRM field.

4.4.3.11 Frame-level DCT AC Coding Set Index

DCTACFRM is a variable-length field that is present in the picture layer if DCTACMBF = 0. This field indexes the coding set used to decode the DCT AC coefficients for the intra- and inter-coded blocks in the frame. Table 20 is used to decode the DCTACFRM field.

4.4.3.12 Intra DCT DC Table

The DCTDCTAB field has the same meaning as the DCTDCTAB field in baseline I pictures. See section 4.1.1.4 for a description.

4.4.3.13 Preprocess Frame - P Frame (PREPROCFRM)

The PREPROCFRM is only signaled when PREPROC is signaled at the sequence level.

When PREPROCFRM is signaled for the current P Frame, we have to scale up the current decoded frame prior to display, similar to I Frame, while keeping the current reconstructed frame intact. Let Y, U, V denote the YUV planes of the output frame. We scale them up according to the following formula:

$$Y_p[n] = (Y[n] - 128) << 1 + 128;$$

$$U_p[n] = (U[n] - 128) << 1 + 128;$$

$$V_p[n] = (V[n] - 128) << 1 + 128;$$

In addition, we need to scale the previous reconstructed frame prior to using it for motion compensation if the current frame and previous frame are operating at different range. More specifically, there are two cases that require scaling the previous reconstructed frame.

- Current frame's PREPROCFRM is signaled and the previous frame's PREPROCFRM is not signaled. In this case, we need to scale down the previous reconstructed frame as follows:

$$Y_p[n] = (Y[n] - 128) << 1 + 128;$$

$$U_p[n] = (U[n] - 128) << 1 + 128;$$

$$V_p[n] = (V[n] - 128) << 1 + 128;$$

- Current frame's PREPROCFRM is not signaled and the previous frame's PREPROCFRM is signaled. In this case, we need to scale up the previous reconstructed frame as follows:

$$Y_p[n] = (Y[n] - 128) >> 1 + 128;$$

$$U_p[n] = (U[n] - 128) >> 1 + 128;$$

$$V_p[n] = (V[n] - 128) >> 1 + 128;$$

4.4.4 Macroblock Layer Decode

4.4.4.1 Macroblock Types

Macroblocks in P pictures can be one of 3 possible types: 1MV, 4MV, and Skipped. The macroblock type is indicated by a combination of picture and macroblock layer fields. The following sections describe each type and how they are signaled.

1MV Macroblocks

1MV macroblocks can occur in 1-MV and Mixed-MV P pictures. A 1MV macroblock is one where a single MVDATA field is associated with all blocks in the macroblock. The MVDATA field signals whether the blocks are coded as Intra or Inter type. If they are coded as Inter then the MVDATA field also indicates the motion vector differential. See section 4.4.5.1 for a description of how to decode Intra blocks in P pictures and see section 4.4.5.2 for a description of how to decode Inter blocks.

If the P picture is of type 1MV then all the macroblocks in the picture are of type 1MV so there is no need to individually signal the macroblock type.

If the P picture is of type Mixed-MV then the macroblocks in the picture can be of type 1MV or 4MV. In this case the macroblock type (1MV or 4MV) is signaled in the MVTYPEMB field in the picture layer. See section 4.4.3.3 for a description of how the MVTYPEMB field signals the 1MV/4MV macroblock type.

4MV Macroblocks

4MV macroblocks can only occur in Mixed-MV P pictures. A 4MV macroblock is indicated by signaling that the macroblock is 4-MV in the MVTYPEMB picture layer field. Individual blocks within a 4MV macroblock can be coded as Intra blocks. For the 4 luminance blocks, the Intra/Inter state is signaled by the BLKMVDATA field associated with that block. The CBPCY field that indicates which blocks have BLKMVDATA fields present in the bitstream. See section 4.4.4.2 for a description of how the CBPCY field is used in 4MV macroblocks.

The Inter/Intra state for the chroma blocks is derived from the luminance Inter/Intra states. If 2 or more of the luminance blocks are coded as Intra then the chroma blocks are also coded as Intra.

Skipped Macroblocks

Skipped macroblocks can occur in 1-MV, and Mixed-MV P pictures. In all cases, a skipped macroblock is signaled by the SKIPMB bitplane field in the picture layer. See section 4.4.3.4 for a description of the SKIPMB field.

4.4.4.2 Macroblock Decoding Process

The following sections describe the macroblock layer decoding process for P picture macroblocks.

Refer to section 4.4.5.2 for a description of the inverse quantization process.

Decoding Motion Vector Differential

The MVDATA or BLKMVDATA fields encode motion information for the blocks in the macroblock. 1MV macroblocks have a single MVDATA field, and 4MV macroblocks can have between zero and four BLKMVDATA fields (see section 4.4.4.2 for a description of how the CBPCY field is used to encode the number of MVDATA fields in 4MV macroblocks).

Each MVDATA or BLKMVDATA field in the macroblock layer jointly encodes three things: 1) the horizontal motion vector differential component, 2) the vertical motion vector differential component and 3) a binary flag indicating whether any DCT coefficients are present. Whether the macroblock (or block for 4MV) is Intra or Inter-coded is coded as one of the horizontal/vertical motion vector possibilities.

The MVDATA or BLKMVDATA field is a variable length Huffman codeword followed by a fixed length codeword. The value of the Huffman codeword determines the size of the fixed length codeword. The MVTAB field in the picture layer specifies the Huffman table used to decode the variable sized codeword.

The following psuedocode illustrates how the motion vector differential, Inter/Intra type and last-flag information are decoded.

The values: **last_flag, intra_flag, dmv_x** and **dmv_y** are computed in the following pseudocode. The values are defined as follows:

last_flag: binary flag indicating whether any DCT coefficients are present (0 = coefficients present, 1 = coefficients not present)

intra_flag: binary flag indicating whether the block or macroblock is intra-coded (0 = inter-coded, 1 = intra-coded)

dmv_x: differential horizontal motion vector component

dmv_y: differential vertical motion vector component

k_x, k_y: fixed length for long motion vectors

k_x and **k_y** depend on the motion vector range as defined by the MVRANGE symbol (section 3.2.1.8) according to Table 52.

<p align="center">**Table 52: k_x and k_y specified by MVRANGE**</p>

MVRANGE	k_x	k_y	range_x	range_y
0 (default)	9	8	256	128
10	10	9	512	256
110	12	10	2048	512
111	13	11	4096	1024

The value **halfpel_flag** used in the following pseudocode is a binary value indicating whether half-pel or quarter-pel precision is used for the picture. The value of **halfpel_flag** is determined by the picture layer field MVMODE (see section 4.4.3.3). If MVMODE specifies the mode as 1MV or Mixed MV then **halfpel_flag** = 0 and quarter-pel precision is used. If MVODE specifies the mode as 1MV Half-pel or 1MV Half-pel Bilinear then **halfpel_flag** = 1 and half-pel precision is used.

The tables **size_table** and **offset_table** are arrays used in the following pseudocode and are defined as follows:
size_table[6] = {0, 2, 3, 4, 5, 8}
offset_table[6] = {0, 1, 3, 7, 15, 31}

```
index = vlc_decode()     // Use the Huffman table indicated by MVTAB in the picture layer
index = index + 1
if (index >= 37)
{
    last_flag = 1
    index = index - 37
}
else
    last_flag = 0

intra_flag = 0
if (index == 0)
{
    dmv_x = 0
    dmv_y = 0
```

```
}
else if (index == 35)
{
      dmv_x = get_bits(k_x – halfpel_flag)
      dmv_y = get_bits(k_y – halfpel_flag)
}
else if (index == 36)
{
      intra_flag = 1
      dmv_x = 0
      dmv_y = 0
}
else
{
      index1 = index % 6
      if (halfpel_flag == 1 && index1 == 5)
            hpel = 1
      else
            hpel = 0
      val = get_bits (size_table[index1] - hpel)
      sign = 0 - (val & 1)
      dmv_x = sign ^ ((val >> 1) + offset_table[index1])
      dmv_x = dmv_x - sign

      index1 = index / 6
      if (halfpel_flag == 1 && index1 == 5)
            hpel = 1
      else
            hpel = 0
      val = get_bits (size_table[index1] - hpel)
      sign = 0 - (val & 1)
      dmv_y = sign ^ ((val >> 1) + offset_table[index1])
      dmv_y = dmv_x - sign
}
```

Motion Vector Predictors

Motion vectors are computed by adding the motion vector differential computed in the previous section to a motion vector predictor. The predictor is computed from three neighboring motion vectors. The following sections describe how the predictors are calculated for macroblocks in 1MV P pictures, and Mixed-MV P pictures.

Motion Vector Predictors In 1MV P Pictures

Figure 54 shows the three motion vectors used to compute the predictor for the current macroblock. As the figure shows, the predictor is taken from the left, top and top-right macroblocks, except in the case where the macroblock is the last macroblock in the row. In this case, Predictor B is taken from the top-left macroblock instead of the top-right.

Not last macroblock in macroblock row   Last macroblock in macroblock row
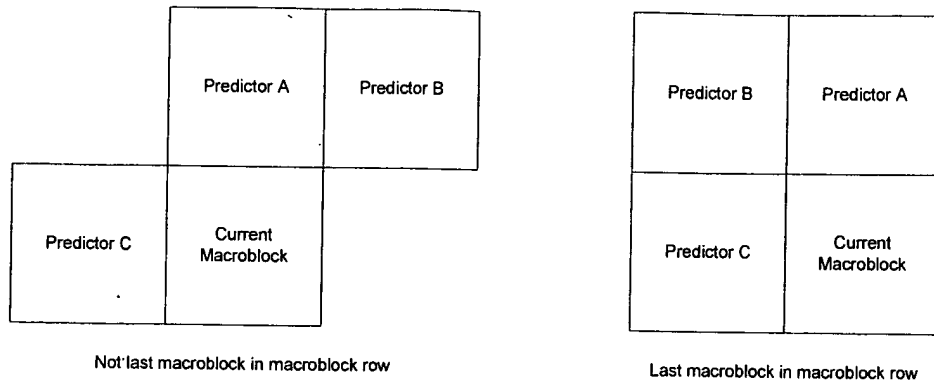
Figure 54: Candidate Motion Vector Predictors in 1MV P Pictures

Motion Vector Predictors In Mixed-MV P Pictures

Figure 55 and Figure 56 show the 3 candidate motion vectors for 1MV and 4MV macroblocks in Mixed-MV P pictures. In the following figures, the larger rectangles are macroblock boundaries and the smaller rectangles are block boundaries.

Not last macroblock in macroblock row   Last macroblock in macroblock row
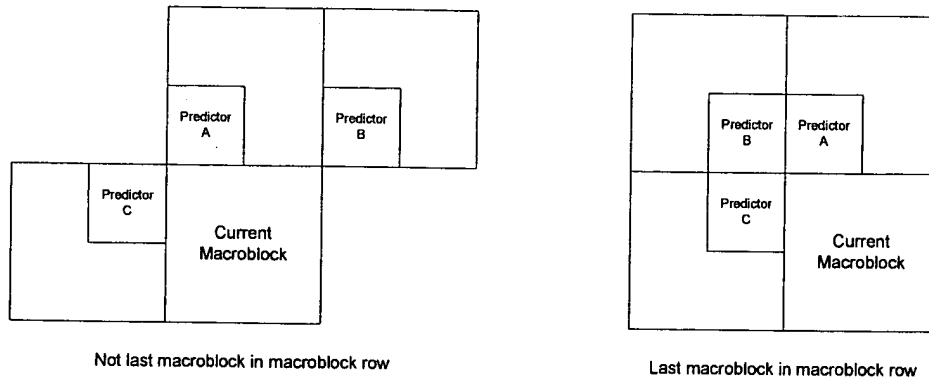
Figure 55: Candidate Motion Vectors for 1MV Macroblocks in Mixed-MV P Pictures

Figure 55 shows the candidate motion vectors for 1MV macroblocks. The neighboring macroblocks may 1MV or 4 MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4MV (ie, predictor A is the motion vector for block 2 in the macroblock above the current and predictor C is the motion vector for block 1 in the macroblock immediately to the left of the current). If any of the neighbors are 1MV macroblocks then the motion vector predictors shown in Figure 55 are taken to be the vectors for the entire macroblock. As the figure shows, if the macroblock is the last macroblock in the row then Predictor B is from block 3 of the top-left macroblock instead of from block 2 in the top-right macroblock as is the case otherwise.

Predictors for
Block 0

Predictors for Block 1 if not
last macroblock in row

Predictors for Block 1 if last
macroblock in row

Predictors for
Block 2

Predictors for
Block 3
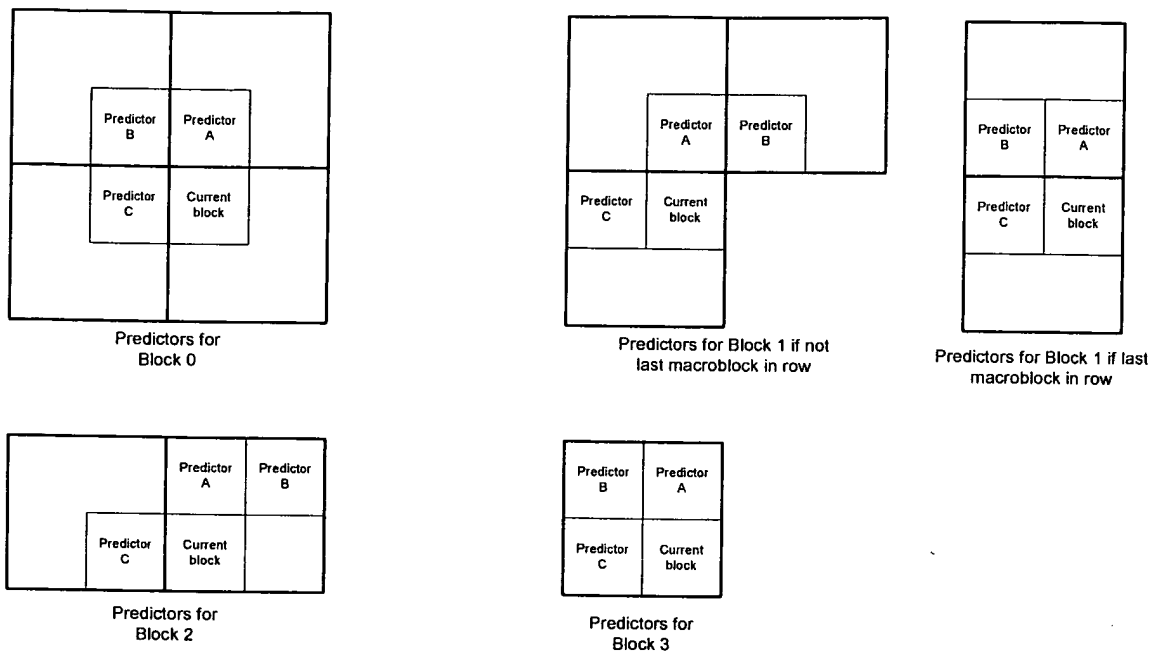
Figure 56: Candidate Motion Vectors for 4MV Macroblocks in Mixed-MV P Pictures

Figure 56 shows the predictors for each of the 4 luminance blocks in a 4MV macroblock. For the case where the macroblock is the last macroblock in the row then Predictor B for block 1 is handled differently. In this case, the predictor is taken from block 2 in the macroblock immediately above the current macroblock instead of from block 2 from the top-left macroblock as is the case otherwise.

Calculating the Motion Vector Predictor

Given the 3 motion vector predictor candidates, the following pseudocode illustrates the process for calculating the motion vector predictors.

```
if (predictorA is out of bounds && predictor C is out of bounds)
{
    predictor_x = 0
    predictor_y = 0
}
else
{
    if (predictorC is out of bounds)
    {
        if (predictorA is intra)
        {
            predictor_x = 0
            predictor_y = 0
        }
        else
        {
            // use top predictor
            predictor_x = predictorA_x
            predictor_y = predictorA_y
        }
```

```
    }
    else if (predictorA is out of bounds)
    {
        if (predictorC is intra)
        {
            predictor_x = 0
            predictor_y = 0
        }
        else
        {
            // use left predictor
            predictor_x = predictorC_x
            predictor_y = predictorC_y
        }
    }
    else
    {
        if (predictorB is out of bounds)
        {
            // set predictorB to inter type and motion vectors to zero
            set predictorB to inter
            predictorB_x = 0
            predictorB_y = 0
        }

        num_intra = 0
        if (predictorA is intra)
        {
            predictorA_x = 0
            predictorA_y = 0
            num_intra = num_intra + 1
        }
        if (predictorB is intra)
        {
            predictorB_x = 0
            predictorB_y = 0
            num_intra = num_intra + 1
        }
        if (predictorC is intra)
        {
            predictorC_x = 0
            predictorC_y = 0
            num_intra = num_intra + 1
        }

        if (num_intra > 1)
        {
            predictor_x = 0
            predictor_y = 0
        }
```

```
        else
        {
            // calculate predictor from A, B and C predictor candidates
            predictor_x = median3(predictorA_x, predictorB_x, predictorC_x)
            predictor_y = median3(predictorA_y, predictorB_y, predictorC_y)
        }
    }
}
```

See section 1.1.8 for the definition of median3.

Hybrid Motion Vector Prediction

If the P picture is 1MV or Mixed-MV then the motion predictor calculated in the previous section is tested relative to the A and C predictors to see if the predictor is explicitly coded in the bitstream. If so, then a bit is decoded that indicates whether to use predictor A or predictor C as the motion vector predictor. The following pseudocode illustrates hybrid motion vector prediction decoding. Hybrid motion vector prediction is not used in field-coded P pictures.

The variables are defined as follows in the pseudocode:

predictor_pre_x: The horizontal motion vector predictor as calculated in the above section
predictor_pre_y: The vertical motion vector predictor as calculated in the above section
predictor_post_x: The horizontal motion vector predictor after checking for hybdrid motion vector prediction
predictor_post_y: The vertical motion vector predictor after checking for hybdrid motion vector prediction

```
if ((predictorA is out of bounds) || (predictorC is out of bounds))
{
    predictor_post_x = predictor_pre_x
    predictor_post_y = predictor_pre_y
}
else
{
    if (predictorA is intra)
        sum = abs(predictor_pre_x) + abs(predictor_pre_y)
    else
        sum = abs(predictor_pre_x - predictorA_x) + abs(predictor_pre_y - predictorA_y)
    if (sum > 32)
    {
        // read next bit to see which predictor candidate to use
        if (get_bits(1) == 0)       // HYBRIDPRED field
        {
            // use top predictor
            predictor_post_x = predictorA_x
            predictor_post_y = predictorA_y
        }
        else
        {
            // use left predictor
            predictor_post_x = predictorC_x
            predictor_post_y = predictorC_y
        }
```

```
        }
        else
        {
            if (predictorC is intra)
                sum = abs(predictor_pre_x) + abs(predictor_pre_y)
            else
                sum = abs(predictor_pre_x – predictorC_x) + abs(predictor_pre_y – predictorC_y)
            if (sum > 32)
            {
                // read next bit to see which predictor candidate to use
                if (get_bits(1) == 0)
                {
                    // use top predictor
                    predictor_post_x = predictorA_x
                    predictor_post_y = predictorA_y
                }
                else
                {
                    // use left predictor
                    predictor_post_x = predictorC_x
                    predictor_post_y = predictorC_y
                }
            }
        }
    }
}
```

Motion Vector Predictors in Skipped Macroblocks

If a macroblock is coded as skipped then the predicted motion vector computed as described above is used as the motion vector for the block, macroblock or field. The block, field or macroblock referenced by the motion vector is used as the current block or macroblock in the current picture. A single bit may be present in the macroblock layer indicating which of the predictor candidates to use.

Reconstructing Motion Vectors

The following sections describe how to reconstruct the luminance and chroma motion vectors for 1MV and 4MV macroblocks.

Luminance Motion Vector Reconstruction

In all cases (1MV and 4MV macroblocks) the luminance motion vector is reconstructed by adding the differential to the predictor as follows:

$$mv\_x = (dmv\_x + predictor\_x) \text{ smod } range\_x$$

$$mv\_y = (dmv\_y + predictor\_y) \text{ smod } range\_y$$

The modulus operation "smod" is a signed modulus, defined as follows:

$$A \text{ smod } b = ((A + b) \% 2 b) - b$$

ensures that the reconstructed vectors are valid. (A smod b) lies within –b and b – 1. range_x and range_y depend on MVRANGE and are specified in Table 52.

Following are the notes about luminance motion vectors in 1MV and 4MV macroblocks.

1MV Macroblock Notes

In 1MV macroblocks there will be a single motion vector for the 4 blocks that make up the luminance component of the macroblock.

If dmv_x decodes to indicate that the macroblock is Intra-coded (as described in the section "Decoding Motion Vector Differential" above) then no motion vectors are associated with the macroblock.

If the SKIPMB field in the picture layer indicates that the macroblock is skipped then dmv_x = 0 and dmv_y = 0 (mv_x = predictor_x and mv_y = predictor_y).

4MV Macroblock Notes

Each of the Inter-coded luminance blocks in a macroblock will have its own motion vector. Therefore there will be between 0 and 4 luminance motion vectors in each 4MV macroblock.

A non-coded block in 4MV macroblocks can occur in one of two ways: 1) if the SKIPMB field in the picture layer indicates that the macroblock is skipped and the MVTYPEMB field in the picture layer indicates that the macroblock is 4MV. All blocks in the macroblock are skipped in this case, or 2) if the CBPCY field (described in the next section) in the macroblock indicates that the block is non-coded. If a block is not coded then dmv_x = 0 and dmv_y (mv_x = predictor_x and mv_y = predictor_y).

Chroma Motion Vector Reconstruction

The chroma motion vectors are derived from the luminance motion vectors. Also, for 4MV macroblocks, the decision of whether to code the chroma blocks as Inter or Intra is made based on the status of the luminance blocks or fields. The following sections describe how to reconstruct the chroma motion vectors for 1MV and 4MV macroblocks. In the sections below cmv_x and cmv_y denote the chroma motion vector components and lmv_x and lmv_y denote the luminance motion vector components.

1MV Chroma Motion Vector Reconstruction

The chroma motion vectors are derived from the luminance motion vectors as follows:

$$cmv\_x = lmv\_x / 2$$

$$cmv\_y = lmv\_y / 2$$

4MV Chroma Motion Vector Reconstruction

The following pseudocode illustrates how the chroma motion vectors are derived from the motion information in the 4 luminance blocks in 4MV macroblocks.

```
if (all 4 luminance blocks are Inter-coded)
{
        // lmv0_x, lmv0_y is the motion vector for block 0
        // lmv1_x, lmv1_y is the motion vector for block 1
        // lmv2_x, lmv2_y is the motion vector for block 2
        // lmv3_x, lmv3_y is the motion vector for block 3
        cmv_x = median4(lmv0_x, lmv1_x, lmv2_x, lmv3_x) / 2
        cmv_y = median4(lmv0_y, lmv1_y, lmv2_y, lmv3_y) / 2
}
else if (3 of the luminance blocks are Inter-coded)
{
        // lmv0_x, lmv0_y is the motion vector for the first Inter-coded block
        // lmv1_x, lmv1_y is the motion vector for the second Inter-coded block
        // lmv2_x, lmv2_y is the motion vector for the third Inter-coded block
        cmv_x = median3(lmv0_x, lmv1_x, lmv2_x) / 2
        cmv_y = median3(lmv0_y, lmv1_y, lmv2_y) / 2
}
else if (2 of the luminance blocks are Inter-coded)
{
        // lmv0_x, lmv0_y is the motion vector for the first Inter-coded block
```

```
// Imv1_x, Imv1_y is the motion vector for the second Inter-coded block
cmv_x = ((Imv0_x + Imv1_x) / 2) / 2
cmv_y = ((Imv0_y + Imv1_y) / 2) / 2
}
else
        Chroma blocks are coded as Intra
```

See section 1.1.8 for the definition of median3 and median4.

Chroma Rounding and Filtering

We use a sequence level 1-bit field that controls the subpixel interpolation and rounding of chroma motion vectors. If the bit is set, then the chroma motion vectors that are at quarter pel offsets will be rounded to the nearest full pel positions and bilinear filtering will be used for all chroma interpolation. The purpose of this mode is speed optimization of the decoder.

The motivation for this optimization is the significant difference between the complexities of interpolating pixel offsets that are at a) integer pel; b) half pel; c) at least one coordinate (of x and y) at a quarter pel; and d) both coordinates at quarter pel positions. The ratio of a:b:c:d is roughly 1:4:4.7:6.6. By applying this mode we can favor a) and b), thus cutting down on decoding time. Since this is being done only for chroma interpolation, the coding and quality loss (especially visible quality) are both negligible.

If the sequence level bit FASTUVMC = 1, then a final level of rounding is done on the chroma motion vectors as follows –

```
if (FASTUVMC)
{
        // RndTbl[-3] = -1, RndTbl[-2] = 0, RndTbl[-1] = +1, RndTbl[0] = 0
        // RndTbl[1] = -1, RndTbl[2] = 0, RndTbl[3] = +1
        cmv_x =       cmv_x + RndTbl[cmv_x % 4];
        cmv_y =       cmv_y + RndTbl[cmv_y % 4];
}
```

In the above, cmv_x and cmv_y represent the x and y coordinates of the chroma motion vector in units of quarter pels. % represents the modulus (or remainder) operation, which is defined thus: $(x \% a) = -(-x \% a)$, i.e. the modulus of a negative number is equal to the negative of the modulus of the corresponding positive number. Thus, when cmv_x (or cmv_y) is divisible by 4 then we have an integer offset; when cmv_x % 4 = +/-2 then we have a half pel offset, and when cmv_x % 4 = +/-1 or +/-3 we have a quarter pel offset. As can be seen by the above re-mapping operation, the quarter pel positions are being disallowed by rounding the chroma motion vector to the nearest integer position (half pel positions are left unaltered).

This forces the chroma co-ordinates to be remapped to integer and half pel positions. Furthermore, bilinear filtering is used for all chroma interpolations if FASTUVMC = 1 for further speedup.

Coded Block Pattern

Figure 18 shows the position of the CBPCY field within the P picture macroblock layer. The CBCPY field is a variable-length code that decodes to a 6-bit field.

The Huffman codetable used to decode CBPCY is specified by the CBPTAB field in the picture layer. See section 4.4.3.6 for a description of the CBPTAB field.

The CBPCY field is used differently depending on whether the macroblock is 1MV or 4MV. The following sections describe how CBPCY is used in each macroblock type.

CBPCY in 1MV Macroblocks

The CBPCY field is present in the 1MV macroblock layer if:

1) The MVDATA field indicates that the macroblock is Inter-coded and,

2) The MVDATA field indicates that at least one block contains coefficient information. This is indicated by the 'last' value decoded from MVDATA. See section 4.4.4.2 for a description of MVDATA decoding.

If the CBPCY field is present then it decodes to a 6-bit field indicating which of the blocks contain at least one non-zero coefficient. Figure 57 shows how the CBPCY bitfield corresponds to the block numbers.

5	4	3	2	1	0

Figure 57: Bit-position/block correspondence for CBPCY

A '1' in one of the positions indicates that the corresponding block has at least one non-zero AC coefficient if the macroblock is Intra-coded or at least one non-zero DC or AC coefficient if the macroblock is Inter-coded.

A '0' in one of the positions indicates that the corresponding block does not contain any non-zero AC coefficients if the macroblock is Intra-coded or any non-zero DC or AC coefficients if the macroblock is Inter-coded.

CBPCY in 4MV Macroblocks

The CBPCY field is always present in the 4MV macroblock layer. The CBPCY bit positions for the luminance blocks (bits 0-3) have a slightly different meaning than the bit positions for chroma blocks (bits 4 and 5).

For the luminance blocks:

A '0' indicates that the corresponding block does not contain motion vector information or any non-zero coefficients. In this case, the BLKMVDATA field is not present for that block and the predicted motion vector is used as the motion vector and there is no residual data. If the motion vector predictors indicate that hybrid motion vector prediction is used then a single bit is present indicating the motion vector predictor candidate to use. Refer to section 4.4.4.2 for a description of computing the motion vector predictor.

A '1' indicates that the BLKMVDATA field is present for the block. The BLKMVDATA field indicates whether the block is Inter or Intra-coded and whether there is coefficient data for the block. If it is Inter coded, the BLKMVDATA field also contains the motion vector differential. If the 'last flag' decoded from BLKMVDATA (described in section 4.4.4.2) decodes to 1 then no AC coefficient information is present if the block is Intra-coded or no DC or AC coefficient is present if the block is Inter-coded. If 'last flag' decodes to 0 then there is at least one non-zero AC coefficient if the block is Intra-coded or at least one non-zero DC or AC coefficient if the block is Inter-coded.

For the chroma blocks:

A '0' indicates that the block does not contain any non-zero AC coefficients if the block is Intra-coded or any non-zero DC or AC coefficients if the block is Inter-coded.

A '1' indicates that the corresponding block has at least one non-zero AC coefficient if the block is Intra-coded or at least one non-zero DC or AC coefficient if the block is Inter-coded.

Macroblock-level DCT Coding Set Index

The DCTTAB field is present if the picture level field DCTACMBF = 1 (see section 4.4.3.10). This indicates that macroblock-level coding set selection is enabled. DCTTAB is a variable-length code that specifies a coding set index. The index is used to select the coding set used to decode the AC coefficient information in intra blocks or the DC and AC coefficient information in inter blocks. Refer to section 4.4.5.1 for a description of AC coefficient decoding in Intra blocks and section 4.4.5.2 for Inter blocks. Table 20 shows the code-table used to decode the DCTTAB field.

MB-level Transform Type

The TTMB field is present only in Inter macroblocks. As described in section 3.2.2.10 TTMB encodes the transform type, the signaling mode and the transform subblock pattern.

If the signaling mode is macroblock signaling then the transform type decoded from the TTMB field is the same for all blocks in the macroblock. If the transform type is 8x4 or 4x8 then a subblock pattern is also decoded from the TTMB field. In this case, the subblock pattern applies to the first coded block in the macroblock. If the transform type is 4x4 then the subblock pattern in encoded in the SUBBLKPAT field at the block level. If the transform type

is 8x4 or 4x8 then the subblock patterns for all the blocks after the first one are coded in the SUBBLKPAT field at the block level.

If the signaling mode is block signaling then the transform type decoded from the TTMB field is applied to the first coded block in the macroblock and the TTBLK field is not present for the first coded block. For the remaining coded blocks, the TTBLK field indicates the transform type for that block. If TTMB field indicates that the first transform type is 8x4 or 4x8 then a subblock pattern is also decoded from the TTMB field. In this case, the subblock pattern applies to the first coded block in the macroblock.

4.4.5 Block Layer Decode

4.4.5.1   Intra Coded Block Decode

The process for decoding Intra blocks in P pictures is similar to the process for decoding Intra blocks in baseline I picture as described in section 4.1.3 with the following differences.

Coefficient Scaling

For DC and AC prediction, the coefficients in the predicted blocks are scaled if the macroblocks quantizers are different than that of the current block. The scaling process is the same as decribed in 4.2.3.8.

AC Prediction in Intra blocks in 4MV or Field-coded macroblocks.

Refer to section 4.1.3.7 for a description of AC prediction. AC prediction in Intra-coded blocks within 4MV macroblocks is similar except that bitstream information is present within the block that controls whether AC prediction is used and which predictor candidate to use. Figure 24 shows the location of the AC prediction information (the ACPREDBLK field) in the Intra block bitstream. The following sections describe how AC prediction is performed in 4MV or Field-coded macroblocks.

If the top predictor is selected then the top row of AC coefficients from the block above the current block are used as the predictors for the top row of AC coefficients from the current block. If the left predictor is selected then the first column of AC coefficients from the block to the left of the current block are used as the predictors for the left column of AC coefficients from the current block. The pseudocode below illustrates the process for deciding the AC predictors in Intra blocks in 4MV or Field-coded macroblocks.

In the pseudocode, predictorA is the block immediately above the current block, predictorC is the block immediately to the left of the current block and predictorB is the block immediately above and to the left of the current block. The result of the pseudocode is that the variable *use_ac_prediction* determines whether AC prediction is used and *prediction_direction* determines which block is used as the predictor.

Note that in the following pseudocode, the coefficients in the predictor blocks are scaled if the macroblock quantizer scales are different. The previous section describes the scaling operation.

```
use_ac_prediction = FALSE
if ((predictorA is Intra) && (predictorC is Intra))
{
    if (abs(predictorB's DC coefficient – predictorC's DC coefficient) <
        abs(predictorB's DC coefficient – predictorA's DC coefficient))
    {
        prediction_direction = UP
    }
    else
        prediction_direction = LEFT

    if (get_bits(1) == 0)      // ACPREDBLK field
        use_ac_prediction = TRUE
    else
    {
        if (get_bits(1) == 0)       // ACPREDBLK field
        {
```

```
            if (prediction_direction == UP)
                  prediction_direction = LEFT
            else
                  prediction_direction = UP
            use_ac_prediction = TRUE
      }
   }
}
else if ((predictorA is Intra) || (predictorC is Intra))
{
   if (predictorA is Intra)
         prediction_direction = UP
   else
         prediction_direction = LEFT

   if (get_bits(1) == 0)       // ACPREDBLK field
         use_ac_prediction = TRUE
}
```

AC Prediction in Intra blocks in 1MV macroblocks

AC prediction in Intra blocks within 1MV macroblocks is the same as Intra blocks in I pictures as described in section 4.1.3.7. The exception is if the top predictor block and left predictor block are not Intra-coded then AC prediction is not used, even if ACPRED = 1 in the macroblock layer. If just one of the predictors is Intra coded (either the top or the left) then it is used as the predictor. If both are Intra-coded then the method described in section 4.1.3.7 is used. In this case, if the top-left block is not Intra then the DC value is assumed to be 0.

Zig-zag Scan

The zig-zag scan order used to scan the run-length decoded DCT coefficients into the 8x8 array is the same as that used for the 8x8 Inter block as described in section 4.4.5.2. This differs from Intra blocks in I pictures which use one of 3 zig-zag scans depending on the prediction direction.

Coding Sets

If the coding set used to decode the AC coefficients is signaled at the frame level, then the DCTACFRM field is used to specify the coding set index used for decoding the Y and Cr/Cb AC coefficients (see section 4.1.3.4 for a description of the AC coding sets). The index decoded from the DCTACFRM field is used to select the intra coding set used to decode the Y blocks and is used to select the inter coding set used to decode the Cr/Cb blocks. This differs from the process used for I pictures where the DCTACFRM specifies the index for the inter coding set and the DCTACFRM2 field specifies the index for the intra coding set. The P picture header does not contain the DCTACFRM2 field. The correspondence between the coding set index and the coding set depends on the the value of PQINDEX. Tables Table 53 and Table 54 below show the correspondence for PQINDEX <= 7 and PQINDEX > 7. Section 5.5 contains the table information.

Table 53: Index/Coding Set Correspondance for PQINDEX <= 7

	Y blocks	Cr and Cb blocks
Index	Table	Table
0	High Rate Intra	High Rate Inter
1	High Motion Intra	High Motion Inter
2	MPEG-4 Intra	MPEG-4 Inter

Table 54: Index/Coding Set Correspondance for PQINDEX > 7

	Y blocks	Cr and Cb blocks
Index	Table	Table
0	Low Motion Intra	Low Motion Inter
1	High Motion Intra	High Motion Inter
2	MPEG-4 Intra	MPEG-4 Inter

4.4.5.2   Inter Coded Block Decode

Figure 58 illustrates the process for reconstructing Inter blocks. For illustration the figure shows the reconstruction of a block whose 8x8 error signal is coded with two 8x4 DCTs. The 8x8 error block can also be transformed with two 4x8 DCTs or one 8x8 DCT. The steps required to reconstruct an inter-coded block include: 1) transform type selection, 2) sub-block pattern decode, 3) coefficient decode, 4) inverse DCT, 5) obtain predicted block and 6) motion compensation (add predicted and error blocks). The following sections describe these steps.
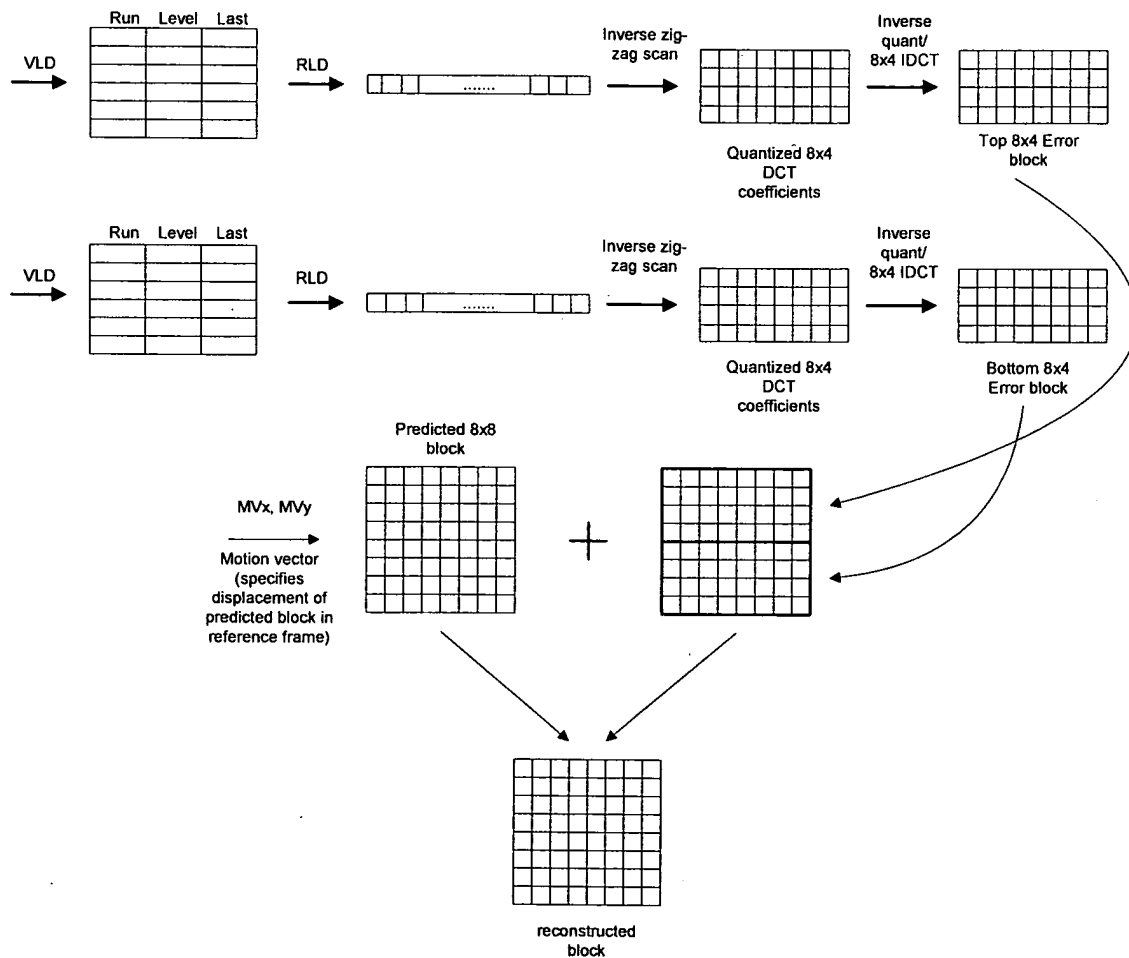
Figure 58: Inter block reconstruction

Transform Type Selection
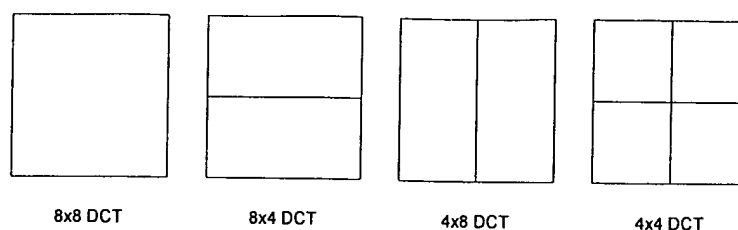
Figure 59: Transform Types

If variable-sized transform coding is enabled (signaled by the sequence-level field VSTRANSFORM = 1 as described in section 3.1.9) then the 8x8 error block can be transformed using one 8x8 DCT, or as shown in Figure 59, divided vertically and transformed with two 8x4 DCTs or divided horizontally and transformed with two 4x8 DCTs or divided into 4 quadrants and transformed with 4 4x4 DCTs. The transform type is signaled at the picture, macroblock or block level. As shown in Tables Table 21, Table 22 and Table 23 if TTMB indicates that the signal level is Block then the transform type is signaled at the block level. If the transform type is specified at the block level then the TTBLK field is present within the bitstream as shown in Figure 25. This field indicates the transform type used for the block. Tables Table 29Table 30Table 31 show the code tables used to encode the transform types if block mode signaling is used.

If variable-sized transform coding is not enabled then the 8x8 DCT is used for all blocks.

Subblock Pattern Decode

If the transform type is 8x4, 4x8 or 4x4 then the decoder needs information about which of the subblocks have non-zero coefficients. For 8x4 and 4x8 transform types, the subblock pattern is decoded as part of the TTMB or TTBLK field. If the transform type is 4x4 then the SUBBLKPAT field is present in the bitstream as shown in Figure 25. Section 3.2.3.16 describes the SUBBLKPAT field.

If the subblock pattern indicates that no non-zero coefficients are present for the subblock then no other information for that subblock is present in the bitstream. For the 8x4 transform type, the data for the top subblock (if present) is coded first followed by the bottom subblock. For the 4x8 transform type, the data for the left subblock (if present) is coded first followed by the right subblock. For the 4x4 transform type, the data for the upper left subblock is coded first followed, in order, by the upper right, lower left and lower right subblocks.

Coefficient Bitstream Decode

The first step in reconstructing the inter-coded block is to reconstruct the DCT coefficients. The process for decoding the bitstream to obtain the run, level and last flags for each non-zero coefficient in the block or sub-block is nearly identical to the process described in section 4.1.3.4 for decoding the AC coefficients in intra blocks. The two differences are:

1) Unlike the decoding process for intra blocks, the DC coefficient is not differentially coded. No distinction is made between the DC and AC coefficients and all coefficients are decoded using the same method.

2) Unlike the decoding process for intra blocks in I pictures (described in section 4.1.3.4) where the Y block coefficients are decoded using one of the three intra coding sets and the Cr and Cb block coefficients are decoded using one of the three inter coding sets, the Y and Cr/Cb inter blocks all use the same inter coding set.

3) The correspondence between the coding set index and the coding set depends on the value of PQINDEX. The following tables show the correspondence for PQINDEX <= 6 and PQINDEX > 6.

Table 55: Index/Coding Set Correspondance for PQINDEX <= 7

Y, Cr and Cb blocks	
Index	Table
0	High Rate Inter
1	High Motion Inter
2	MPEG-4 Inter

Table 56: Index/Coding Set Correspondance for PQINDEX > 7

Y, Cr and Cb blocks	
Index	Table
0	Low Motion Inter
1	High Motion Inter
2	MPEG-4 Inter

Run-level Decode

The process for decoding the run-level pairs obtained in the coefficient decoding process described above is nearly the same as described in section 4.1.3.5. The difference is that because all coefficients are run-level encoded (not just the AC coefficients as in intra blocks) the run-level decode process produces a 16-element array in the case of 4x4 DCT, a 32-element array in the case of 8x4 or 4x8 DCT blocks or a 64-element array in the case of 8x8 DCT blocks.

Zig-zag Scan of Coefficients

The one-dimensional array of quantized coefficients produced in the run-level decode process described above are scanned out into a two-dimensional array in preparation for the IDCT. The process is similar to that described in section 4.1.3.6 for intra blocks. The differences are:

1) Each DCT type has an associated zig-zag scan array. The zig-zag scan arrays for 8x8, 8x4, 4x8 and 4x4 DCT are shown in section 5.6.2.

2) Unlike the zig-zag scanning process for intra blocks where one of three arrays are used depending on the DC prediction direction, only one array is used for inter blocks.

Inverse Quantization

The non-zero quantized coefficients reconstructed as described in the sections above are inverse quantized in one of two ways depending on the value of PQUANT.

If the 3QP deadzone quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep)$$

If the 5QP deadzone quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep) + \text{sign}(quant\_coeff) * quant\_scale$$

where:

quant_coeff is the quantized coefficient

dequant_coeff is the inverse quantized coefficient

quant_scale = The quantizer scale for the block (either PQUANT or MQUANT)

halfstep = The half step encoded in the picture layer as described in section 3.2.1.8.

PQUANT is encoded in the picture layer as described in section 3.2.1.7.

MQUANT is encoded as described in section 4.4.4.2.

Inverse DCT

After reconstruction of the DCT coefficients, the resulting 8x8, 8x4 or 4x8 blocks are processed by the appropriate separable two-dimensional inverse discrete cosine transforms (IDCT). The 8x8 blocks are transformed using the 8x8 IDCT, the 8x4 blocks are transformed using the 8x4 IDCT and the 4x8 blocks are transformed using the 4x8 IDCT. The inverse transforms output ranges from -256 to +255 after clipping to be represented with 9 bits.

The transfer function of the 8x8 inverse transform is given by:

$$f(x,y) = 1/4 \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v) \cos[\pi(2x+1)u/16] \cos[\pi(2y+1)v/16]$$

with $u, v, x, y = 0, 1, 2, \ldots, 7$

where x,y = spatial coordinates in the pixel domain,

        u,v = coordinates in the transform domain,

        $C(u) = 1/\sqrt{2}$ for $u = 0$, otherwise 1,

        $C(v) = 1/\sqrt{2}$ for $v = 0$, otherwise 1.

The transfer function of the 8x4 inverse transform is given by:

$$f(x,y) = 1/\sqrt{8} \sum_{u=0}^{7} \sum_{v=0}^{3} C(u)C(v)F(u,v) \cos[\pi(2x+1)u/16] \cos[\pi(2y+1)v/8]$$

with $u, x = 0, 1, 2, \ldots, 7$

with $v, y = 0, 1, 2, 3$

where x,y = spatial coordinates in the pixel domain,

        u,v = coordinates in the transform domain,

        $C(u) = 1/\sqrt{2}$ for $u = 0$, otherwise 1,

        $C(v) = 1/\sqrt{2}$ for $v = 0$, otherwise 1.

The transfer function of the 4x8 inverse transform is given by:

$$f(x,y) = 1/\sqrt{8} \sum_{u=0}^{3} \sum_{v=0}^{7} C(u)C(v)F(u,v) \cos[\pi(2x+1)u/8] \cos[\pi(2y+1)v/16]$$

with $u, x = 0, 1, 2, 3$

with $v, y = 0, 1, 2, \ldots, 7$

where x,y = spatial coordinates in the pixel domain,

        u,v = coordinates in the transform domain,

        $C(u) = 1/\sqrt{2}$ for $u = 0$, otherwise 1,

        $C(v) = 1/\sqrt{2}$ for $v = 0$, otherwise 1.

NOTE – Within the block being transformed, $x = 0$ and $y = 0$ refer to the pixel nearest the left and top edges of the picture respectively.

See section 4.11 regarding IDCT conformance.

Motion Compensation

The 8x8, 8x4, 4x8 or 4x4 error block or blocks are added to the predicted 8x8 block to produce the reconstructed block. The motion vector decoded in the macroblock header (described in section 4.4.4.2) is used to obtain the predicted block in the reference frame.

The horizontal and vertical motion vector components represent the displacement between the block currently being decoded and the corresponding location in the reference frame. Positive values represent locations that are

below and to the right of the current location. Negative values represent locations that are above and to the left of the current location.

If the picture layer field MVMODE (see section 3.2.1.17) indicates that 1MV Halfpel or 1MV Halfpel Bilinear is used as the motion compensation mode then all motion vectors are expressed in half-pixel resolution. For example, a horizontal motion component of 4 would indicate a postion 2 pixels to the right of the current position and a value of 5 would indicate a position of 2 ½ pixels to the right. If the picture layer field MVMODE (see section 3.2.1.17) indicates that 1MV or Mixed MV is used as the motion compensation mode then all motion vectors are expressed in quarter-pixel resolution. For example, a horizontal motion component of 4 would indicate a postion 1 pixel to the right of the current position and a value of 5 would indicate a position of 1 1/4 pixels to the right. In 1MV Halfpel Bilinear mode, all non-integer pixel motion vector offsets use a bilinear filter to compute the interpolated pixels. Otherwise, all non-integer pixel motion vector offsets use a bicubic filter to compute the interpolated pixels

Bilinear Interpolation

The following sections describe the bilinear filter operations.

The bilinear filter operates as shown in Figure 60.
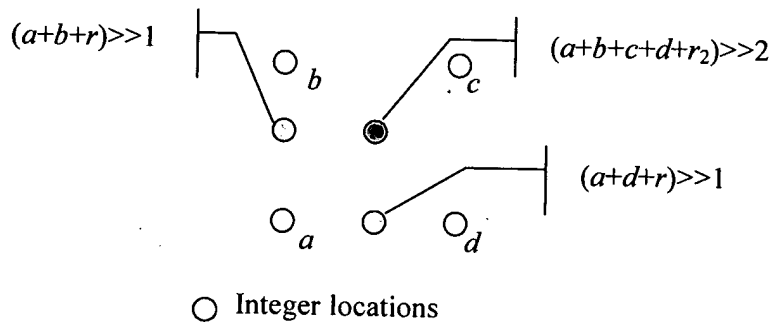
Figure 60: Bilinear filter operation

The value $r = 1 - R$ (and $r_2 = 2 - R$), where R is the frame level rounding control value as described in section 4.4.6.

Bicubic Interpolation

The following sections describe the bicubic filter operations.
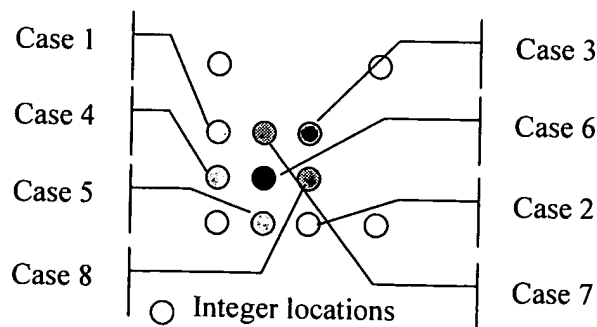
Figure 61: Quarter pel bicubic filter cases

Figure 61 shows all the possible unique interpolated positions. They are:

Case 1: full-pel horizontal, half-pel vertical

Case 2: half-pel horizontal, full-pel vertical

Case 3: half-pel horizontal, half-pel vertical

Case 4: full-pel horizontal, quarter-pel vertical

Case 5: quarter-pel horizontal, full-pel vertical

Case 6: quarter-pel horizontal, quarter-pel vertical

Case 7: half-pel horizontal, quarter-pel vertical

Case 8: quarter-pel horizontal, half-pel vertical

One-dimensional Bicubic Interpolation (Cases 1, 2, 4 and 5)

In Figure 61, cases 1, 2, 4 and 5 represent the cases where interpolation occurs in only one dimension – either horizontal or vertical. The following filters are used for the possible shift locations:

½ pel shift F1: [-1   9   9   -1]

¼ pel shift F2: [-4   53   18   -3]

¾ pel shift F3: [-3   18   53   -4]

Figure 62 shows the pixels that are used to compute the interpolated pixels for each case. S denotes the sub-pixel position. P1, P2, P3 and P4 represent the integer pixel positions. The figure shows horizontal interpolation but the same operation applies to vertical interpolation.

½ **Pixel Shift**

¼ **Pixel Shift**
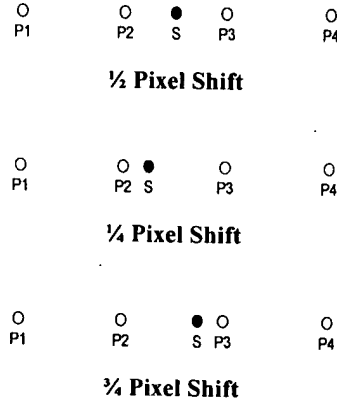
¾ **Pixel Shift**

Figure 62: Pixel Shifts

The following equations show the filtering operation for each case:

$(-1*P1 + 9*P2 + 9*P3 -1*P4 + 8 - r) >> 4$ (1/2 pixel shift)

$(-4*P1 + 53*P2 + 18*P3 - 3*P4 + 32 - r) >> 6$ (1/4 pixel shift)

$(-3*P1 + 18*P2 + 53*P3 -4*P4 +32 - r) >> 6$ (3/4 pixel shift)

The value r in the equations above depends on R, the frame-level round control value (see section 4.4.6 for a description) and the interpolation direction as follows:

$$r = \begin{cases} 1 - R & (vertical\ direction - cases\ 1\ and\ 4) \\ R & (horizontal\ direction - cases\ 2\ and\ 5) \end{cases}$$

Two-dimensional Bicubic Interpolation

In Figure 61, cases 3, 6, 7 and 8 are the cases where interpolation occurs in both the horizontal and vertical directions.

Two-dimensionally interpolated pixel locations first interpolate along the vertical direction, and then along the horizontal direction using the appropriate filter among F1, F2 and F3 specified above. Rounding is applied after

vertical filtering and after horizontal filtering. The rounding rule ensures retention of maximum precision permitted by 16 bit arithmetic in the intermediate results.

The rounding rule after vertical filtering is defined as

$$(S + rndCtrlV) >> shiftV$$

where

S = vertically filtered result, ie -1*P1 + 9*P2 + 9*P3 -1*P4 for ½ pixel shift

$shiftV$ = { 1, 5, 3, 3 } for cases 3, 6, 7 and 8 respectively.

$rndCtrlV = 2^{shiftV-1}$ - 1 + R (see section 4.4.6 for a description of R)

The rounding rule after horizontal filtering is:

$$(S + 64 - R) >> 7.$$

where

S = horizontally filtered result

R = frame level round control value (see section 4.4.6)

All of the bicubic filtering cases can potentially produce an interpolated pixel whose value is negative, or larger than the maximum range (255). In these cases, the output is clipped to lie within the range – underflows are set to 0 and overflows to 255.

Adding Error and Predictor

The 8x8 predicted block is added to the 8x8 error block to form the reconstructed 8x8 block. The pseudo-code in Figure 63 illustrates this process.

```
for (row= 0; row < 8; row++)
{
  for (col = 0; col < 8; col++)
      reconblock[row*8 + col] = clip(predblock[row*8 + col] + errorblock[row*8 + col])
}

where:
clip(n) =
        0 if n < 0
        255 if n > 255
        n otherwise
```

Figure 63: Inter block reconstruction pseudo-code

4.4.6 Rounding Control

Section 4.4.5.2 describes the interpolation operations used to generate subpixel values in the reference blocks. Rounding is controlled by a value R called the rounding control value. The value of R toggles back and forth between 0 and 1 at each P frame. At each I frame, the value of R is reset to 0. Therefore, the value of R for the first P frame following an I frame is 0.

4.4.7 Intensity Compensation

If the picture layer field MVMODE indicates that intensity compensation is used for the frame then the pixels in the reference frame are remapped prior to using them as predictors for the current frame. As section 4.4.3.3 describes, when intensity compensation is used, the LUMSCALE and LUMSHIFT fields are present in the picture bitstream. The following pseudocode illustrates how the LUMSCALE and LUMSHIFT values are used to build the lookup table used to remap the reference frame pixels.

```
if (LUMSCALE == 0)
{
    iScale = - 64
    iShift = 255 * 64 + 32 - LUMSHIFT *2 * 64
}
else {
    iScale = LUMSCALE + 32
    if (LUMSHIFT > 31)
        iShift = LUMSHIFT * 64 - 64 * 64;
    else
        iShift = LUMSHIFT * 64;
    }


// build LUTs
for (i = 0; i < 256; i++)
{
    j = (iScale * i + iShift + 32) >> 6
    if (j > 255)
        j = 255
    else if (j < 0)
            j = 0
    LUTY[i] = j
    j = (iScale * (i - 128) + 128 * 64 + 32) >>6
    if (j > 255)
        j = 255
    else if (j < 0)
        j = 0
    LUTUV[i] = j
}
```

The Y component of the reference frame is remapped using the LUTY[] table generated above and the U and V components are remapped using the LUTUV[] table as follows:

$$\overline{p_Y} = LUTY[p_Y]$$

$$\overline{p_{UV}} = LUTUV[p_{UV}]$$

Where p_Y is the original luminance pixel value in the reference frame and $\overline{p_Y}$ is the remapped luminance pixel value in the referece frame and p_{UV} is the original U or V pixel value in the reference frame and $\overline{p_{UV}}$ is the remapped U or V pixel value in the referece frame.

4.5 Interlace P Frame Decoding

The following sections describe each P picture type.

In Field-coded P pictures, each macroblock can be encoded in field mode or frame mode. In a field-coded macroblock, two motion vectors are associated with the macroblock. In frame mode, one motion vector is associated with the macroblock. The Field P picture mode is signaled by INTERLACE = 1 (sequence layer) and INTRLCF = 1 (picture layer). If the P picture is field-coded then the frame or field mode for each macroblock is indicated by the INTRLCMB bitplane field in the picture layer as described in section 4.5.1.3.

4.5.1 Interlace P Picture Layer Decode

Figure 11 shows the elements that make up the Interlaced P picture layer header. Most of the elements are the same as the Progressive P picture layer. The following sections provide extra detail for some of the elements.

4.5.1.1 Resolution Change

4.5.1.2 Picture Resolution Index

The RESPIC2 field in P pictures specifies the scaling factor of the decoded interlaced P picture relative to a full resolution frame. The decoded picture may be full resolution or half the original resolution in the horizontal dimension. Table 9 shows how the scaling factor is encoded in the RESPIC2 field.

If the picture resolution for the current P frame is different than the resolution of the previous frame then the reference frame must be resampled to match the new frame resolution. The process described in section 4.1.1.6 must be used to resample the reference frame.

4.5.1.3 Interlaced Field/Frame Decoding

If the sequence layer field INTERLACE = 1 then a picture layer field INTRLCF is present in the bitstream. INTRLCF is a 1-bit field that indicates the mode used to code the macroblocks in that frame. If INTRLCF = 0 then all macroblocks in the frame are coded in frame mode. If INTRLCF = 1 then the macroblocks may be coded in field or frame mode and the INTRLCMB field is present in the picture layer. INTRLCMB is a bitplane coded field that indicates the field/frame coding status for each macroblock in the picture. The decoded bitplane represents the interlaced status for each macroblock as a field of 1-bit values in raster scan order from upper left to lower right. Refer to section 4.10 for a description of the bitplane coding. A value of 0 indicates that the corresponding macroblock is coded in frame mode. A value of 1 indicates that the corresponding macroblock is coded in field mode.

4.5.1.4 Picture Layer Intensity Compensation Decoding

The P picture layer contains syntax elements that control the motion compensation mode and intensity compensation for the frame. The INTCOMP field is a one bit value that signals that intensity compensation is used in the frame. If intensity compensation is signaled then the LUMSCALE and LUMSHIFT fields follow in the picture layer. LUMSCALE and LUMSHIFT are 6-bit values which specify parameters used in the intensity compensation process. Refer to section 4.4.7 for a description of intensity compensation decode.

4.5.1.5 Skipped Macroblock Decoding

The P picture layer contains the SKIPMB field which is a bitplane coded field that indicates the skipped/not-skipped status of each macroblock in the picture. The decoded bitplane represents the skipped/not-skipped status for each macroblock as a field of 1-bit values in raster scan order from upper left to lower right. Refer to section 4.10 for a description of the bitplane coding. A value of 0 indicates that the macroblock is not skipped. A value of 1 indicates that the macroblock is coded as skipped. A skipped status for a macroblock means that there is no coded information for that macroblock and the constituent blocks in the bitstream.

4.5.2 Macroblock Layer Decode

4.5.2.1 Macroblock Types

Macroblocks in P pictures can be one of 3 possible types Frame-code and Field-coded and Skipped. The macroblock type is indicated by a combination of picture and macroblock layer fields. The following sections describe each type and how they are signaled.

Frame-coded Macroblocks

A Frame-coded macroblock is one where a single MVDATA field is associated with all blocks in the macroblock. The MVDATA field signals whether the blocks are coded as Intra or Inter type. If they are coded as Inter then the MVDATA field also indicates the motion vector differential.

Field-coded Macroblocks

A Field-coded macroblock is indicated by signaling that the macroblock is field-coded in the INTRLCMB picture layer field. In field-coded macroblocks, a TOPMVDATA field is associated with the top field blocks (i.e. blocks Y_0, Y_1, U_0, V_0) and a second BOTMVDATA field is associated with the bottom field blocks (i.e. blocks Y_2, Y_3, U_1,

V₁). The field MVDATA is send at the first block of each field. More specifically, The TOPMVDATA is send along with block Y₀ and the BOTMVDATA is send along with block Y₂. The TOPMVDATA field indicates whether the top field blocks are Intra or Inter. If they are Inter then the BLKMVDATA field also indicates the motion vector differential for the top field blocks. Likewise, the second BLKMVDATA field signals the Inter/Intra state for the bottom field blocks. The CBPCY field that indicates which fields have field MVDATA fields present in the bitstream.

Skipped Macroblocks

A skipped macroblock is signaled by the SKIPMB bitplane field in the picture layer. See section 4.4.3.4 for a description of the SKIPMB field.

4.5.2.2   Macroblock Decoding Process

The following sections describe the macroblock layer decoding process for P picture macroblocks.

Refer to section 4.5.3.2 for a description of the inverse quantization process.

Decoding Motion Vector Differential

The MVDATA, TOPMVDATA, and BOTMVDATA field are decoded the same way as Progressive P pictures. See section 0 for a description.

Motion Vector Predictors

Motion vectors are computed by adding the motion vector differential computed in the previous section to a motion vector predictor. The predictor is computed using motion vectors from the three neighboring blocks. The following sections describe how the predictors are calculated for macroblocks in Interlace P pictures.

Motion Vector Predictors In Interlace P Pictures

We derive the motion vector predictor for the current MB by using the motion vector(s) of the left, top, and top-right macroblocks. The predictor motion vector candidates are obtained or computed based on the current MB's type. Figure 64 shows the case for frame coded MB and Figure 65 shows the case for field coded MB.
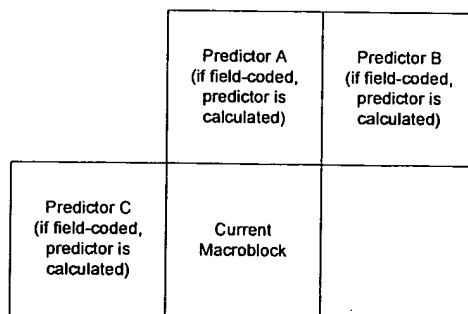
Figure 64: Candidate Motion Vectors for Frame Macroblocks in Interlace P Pictures
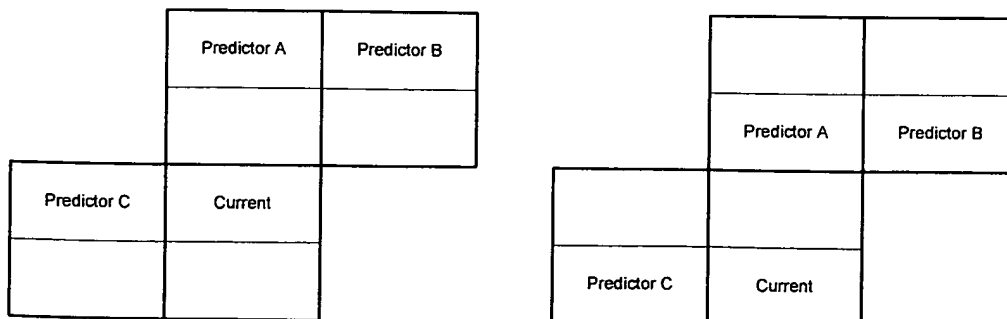
Figure 65: Candidate Motion Vectors for Field Macroblocks in Interlace P Pictures

If the current MB is frame coded, the predictor candidates are computed differently depending on the whether the neighboring MB is frame coded or field coded. If the neighboring MB is frame encoded then its motion vector is taken as the predictor candidate. On the other hand, if the neighboring MB is field encoded then its top and bottom field motion vector are averaged to form the candidate predictor.

If the current MB is field coded, the predictor candidates for the top field is taken neighboring MB's top field and the predictor candidate for the bottom field is taken from the neighboring MB's bottom field. If the neighboring MB is frame coded then its motion vector is taken as both top field predictor and the bottom field predictor.

In both cases, if there are no motion vectors (i.e. frame/field intra MB), the candidate frame/top field/bottom field MV is set to be zero.

Calculating the Motion Vector Predictor

Same as Progressive P Frame as described in section 0

Motion Vector Predictors in Skipped Macroblocks

If a macroblock is coded as skipped then the predicted motion vector computed as described above is used as the motion vector for the block, macroblock or field. If a macroblock is skipped then no information for that macroblock is present in macroblock layer.

Reconstructing Motion Vectors

The following sections describe how to reconstruct the luminance and chroma motion vectors for 1MV, 4MV and Field-coded macroblocks.

Luminance Motion Vector Reconstruction

Same as Progressive P Frame as describe in section 0.

Frame coded Macroblock Notes

There is a single motion vector for the 4 blocks that make up the luminance component of the macroblock.

If dmv_x decodes to indicate that the macroblock is Intra-coded (as described in the section "Decoding Motion Vector Differential" above) then no motion vectors are associated with the macroblock.

If the SKIPMB field in the picture layer indicates that the macroblock is skipped then dmv_x = 0 and dmv_y = 0 (mv_x = predictor_x and mv_y = predictor_y).

Field coded Macroblock Notes

Each one of the two fields in the macroblock will have its own motion vector. Therefore there will be between 0 and 2 luminance motion vectors in each Field-coded macroblock.

A non-coded field in Field-coded macroblocks can occur in one of two ways: 1) if the SKIPMB field in the picture layer indicates that the macroblock is skipped and the INTRLCMB field in the picture layer indicates that the macroblock is Field-coded. All blocks in the macroblock are skipped in this case, or 2) if the CBPCY field (described in the next section) in the macroblock indicates that the field is non-coded. If a field is not coded then dmv_x = 0 and dmv_y (mv_x = predictor_x and mv_y = predictor_y).

Chroma Motion Vector Reconstruction

The chroma motion vectors are derived from the luminance motion vectors. The following sections describe how to reconstruct the chroma motion vectors from the corresponding frame/field motion vector. We note that for frame coded MB, there will be one chrominance motion vector corresponding to the single luminance motion vector. On the other hand, for field coded MB, there will be two chrominance motion vectors, one for the top field and one for the bottom field, corresponding to the two luminance motion vectors. The rules for deriving the chroma vector is the same for both field and frame coded MB and it's only dependent on the luminance motion vector not the type of MB.

Let cmv_x and cmv_y denote the chroma motion vector components and lmv_x and lmv_y denote the corresponding luminance motion vector components. Basically, we will the x component of the chrominance motion vector is scaled by four while the y component of the chrominance motion vector is remains the same

because of the 4:1:1 subsampling. The scaled x component chrominance motion vector is also rounded to a neighboring quarter pixel location.

$frac\_x4 = (lmv\_x << 2) \% 16;$

$int\_x4 = (lmv\_x << 2) - frac\_x;$

$ChromaMvRound [16] = \{0, 0, 0, .25, .25, .25, .5, .5, .5, .5, .5, .75, .75, .75, 1, 1\};$

$cmv\_y = lmv\_y;$

$cmv\_x = Sign (lmv\_x) * (int\_x4 >> 2) + ChromaMvRound [frac\_x4];$

After cmv_x, cmv_y is computed, we check to see if they are out of bound. If they are out of bound then we pull them back as in Progressive P frame coding.

Coded Block Pattern

The CBCPY field is a variable-length code that decodes to a 6-bit field. It and the motion vectors are used to specify whether the block(s) have AC coefficients. We note that for an inter-coded block, all DCT coefficients are considered as AC coefficients whereas for an intra-coded block, the first coefficient is not included as in the set of AC coefficients. The decoding procedure used to decode CBPCY is the same as the Progressive P Picture.

The following sections describe how CBPCY is used in each macroblock type.

CBPCY in Frame coded Macroblocks

The CBPCY field is present in the frame encoded macroblock layer if the 'last' value decoded from MVDATA indicates that there are data following the motion vector decode. If the CBPCY field is present then it decodes to a 6-bit field, one bit for each the four Y blocks, one bit for both U blocks, and one bit for both V blocks. The meaning of the CBPCY is the same as Interlace I pictures.

CBPCY in Field-coded Macroblocks

The CBPCY field is always present in the Field-coded macroblock layer. In this case, the CBPCY and the two field motion vectors are used in conjunction to derive the presence AC coefficients in the blocks of the MB. The meaning of the CBPCY field is exactly the same as the frame-coded macroblocks for bit 1, 3, 4 and 5. That is they indicate the presence of AC coefficients in block Y_1, Y_3, $U_0 + U_1$, and $V_0 + V_1$, respectively. For bit location 0 and 2, the meaning is slightly different as described below:

- A '0' in bit position 0 indicates that the TOPMVDATA field is not present and the motion vector predictor is used as the motion vector for the top field blocks (Y_0, Y_1, U_0, V_0). It also indicates that block Y_0 does not contain any nonzero coefficients.

- A '1' in bit position 0 indicates that the TOPMVDATA field is present. The TOPMVDATA field indicates whether the top field blocks are Inter or Intra-coded. If they are Inter-coded, the TOPMVDATA field also contains the motion vector differential. If the 'last flag' decoded from TOPMVDATA decodes to 1 then no AC coefficients is present for block Y_0 otherwise, there are nonzero AC coefficients for block Y_0.

Similarly, bit position 2 uses the above rule but it uses is BOTMVDATA and it decides whether there are nonzero AC coefficients for block Y_2.

Subblock Pattern U,V

Same as Interlace Intra picture. See section 4.2.2.2.

MB-level Transform Type

Same as Progressive P picture except that it only applies to blocks Y_0, Y_1, Y_2, and Y_3. See section 0 for a description.

4.5.3 Block Layer Decode

4.5.3.1 Intra Coded Block Decode

The process for decoding Intra blocks in Interlace P pictures is the same as the process for decoding Intra blocks in interlace I picture as described in section 4.2.3.

The Coding Sets used is the same as Progressive P pictures. See section 4.4.5.1 for a description.

4.5.3.2 Inter Coded Block Decode

The process for decoding the Inter blocks is similar to the Progressive P pictures with the exception that there are two U and two V blocks of size 4x8.

Transform Type Selection

Same as Progressive P pictures with the exception that the Transform Type Selection is only applied to the luminance blocks. The chrominance blocks are always 4x8 transformed.

Subblock Pattern Decode

Same as Progressive P pictures except for different table. See Table 36.

Motion Compensation

The 8x8, 8x4 or 4x8 error block or blocks are added to the predicted 8x8 block to produce the reconstructed block. The motion vector decoded in the macroblock header is used to obtain the predicted block in the reference frame.

The horizontal and vertical motion vector components represent the displacement between the block currently being decoded and the corresponding location in the reference frame. Positive values represent locations that are below and to the right of the current location. Negative values represent locations that are above and to the left of the current location.

If the current MB is frame encoded, the motion vector is used the same way as the 1 MV MB in Progressive P Picture with bicubic interpolation. On the other hand, if the current MB is field encoded, the top field and bottom field have it's own motion vector. Given a field MV that points to a starting location in the reference frame, we take every other lines starting from the starting location and uses bicubic interpolation to compute the motion prediction for that field.

After the motion prediction for the MB is formed, we add the error to it and clip it to be between 0 and 255.

4.5.4 Intensity Compensation

Same as Progressive P frame except for it is only applied to the Y dimension not the U,V dimension.

4.6   Progressive B Frame Decoding

At the top level, most B frames are coded as bidirectionally predicted frames as the name suggests. In certain cases however it is more economical to code a frame independent of its anchors – in other words as an intra B frame. An intra B frame has the same frame level syntax an inter B frame, but its macro block level decoding follows that of a baseline I frame.

Normal B frames are coded by coding 16x16 tiles of the image (macro blocks). Unlike P frames there is no "4MV" motion compensation mode. At the frame level, only two choices for motion vector resolution are permitted – quarter pel bicubic and half pel bilinear.

4.6.1 Skipped and Dropped Frames

When a B frame is skipped (not coded because it is similar or identical to the previous frame) or dropped (not coded because of bandwidth limitations), it is not sent. The decoder has no knowledge of its existence and continues to display the previous frame.

When an anchor frame is skipped / dropped, there is a possibility that the intermediate B frames may not be skipped / dropped. Without indicating to the decoder that the anchor is skipped / dropped, the decoding process will not be able to function correctly. Consider the following temporal sequence of frames:

I0 B1 P2 B3 P4

The order of transmission is

I0 P2 B1 P4 B3

Now, assume that B1 is identical to I0 and is dropped entirely from the sequence. The transmitted frames are therefore

I0 P2 P4 B3

which are displayed correctly as

I0 P2 B3 P4.

However, if P2 is identical to I0 and B1 is not, and if P2 is dropped from the stream, we have the non-decodable sequence

I0 B1 B3 P4

Hence, it is necessary to indicate to the decoder any anchor frame that is dropped. This is implemented in WMV9 by coding a frame of size 1 byte. This byte is null 0x00.

On the decoder side, it is known that no one byte frame is valid. Thus the decoder identifies skipped P frames and makes a duplicate copy of the existing reference to act as the temporally subsequent reference anchor.

Clearly, there is no dynamic range / resolution / motion vector range change at skipped / dropped P frames. It is worth noting that the fraction coding principle in WMV9 allows for arbitrary dropping of B frames, and indeed variable B frame distance

4.6.2 B Picture Layer Decode

Some B frame specific information is transmitted at the frame level. Apart from the frame type (PTYPE), a symbol called the B frame fraction (BFRACTION) is sent at the frame header. This indicates whether the B frame is coded as Intra, and if not, the scaling factor used to derive the direct motion vectors (explained in section 3.2.1.5). The global states of resolution and range are guaranteed not to change at a B frame – the only permissible points of change are at anchor frames. However, the information relating to resolution and range are transmitted at B frames as well (when these flags are enabled at the sequence layer). The remainder of the section deals only with predicted or "normal" B frames, i.e. those that aren't coded as Intra.

Global luminance change is disallowed at B frames. However, a dummy bit is sent for interlaced content. For progressive B frames, the motion mode is also sent. Only two motion modes are valid – yet there is some redundancy in the way they are encoded.

4.6.2.1   Bitplane Coding

As in P frames, some information is coded as a compressed bitplane that is sent at the frame level. For progressive B frames, two such bitplanes are sent – one denoting skipped macro blocks and the other denoting direct coded macro blocks. This information is sent at the macro block level when the raw coding mode is chosen. See section 3.3 for a description of bitplane coding.

4.6.2.2   Rounding Control

The rounding control parameter used by B frames is identical to that used by the previously decoded anchor.

4.6.2.3   Start Codes

B frames do not contain startcodes.

4.6.3 B Macroblock Layer Decode

Macro blocks in B frames are identified as belonging to one of four modes, viz. *backward, forward, direct* and *interpolated*. The forward mode is akin to conventional P picture prediction. In the forward mode, the B macro block is predicted from its temporally previous anchor frame only. Likewise, backward mode macro blocks are entirely predicted from their temporally subsequent anchor frame.

4.6.3.1   Long and Short Types

When a B frame is closer to its temporally previous reference, it can be expected that the forward coding mode will be used more often. Likewise, when a B frame is closer to the end of its inter-anchor interval, it may be expected

that it references the future anchor more often. This statistical behavior is exploited by flagging the backward and forward mode using two codewords whose interpretation is switched across two sides of the midpoint of the inter-anchor interval.

4.6.3.2 Direct and Interpolated Modes

Direct mode and interpolated mode macro blocks use both the anchors for prediction. Since there are two reference images for these modes, there are two motion vectors for each macro block. The direct mode implicitly derives these motion vectors by appropriately scaling and bounding the motion vectors of the collocated macro block in the temporally subsequent anchor frame. This process is called temporal prediction of motion vectors. The function DirectModeMV( ) derives the direct mode motion vectors (backward : (idbx, idby), forward (idfx, idfy)) from the collocated motion vector (iXMotion, iYMotion) – see reference software for details.

The fraction denoting the temporal position of the current frame within the inter-anchor interval is used as the scaling weight. In theory, it may be possible to play around with this fraction to get some coding benefit. Also, there is no requirement that this fraction actually correspond to the true temporal position – it is certainly possible that the fractions for multiple B frames within two anchors are not monotonic ascending.

When the collocated macro block in the temporally subsequent frame is intra, iXMotion and iYMotion are set to zero. When the same is coded as 4MV, iXMotion and iYMotion are derived from the corresponding chrominance motion vectors. Function ChromaMV( ) in the reference software implements this.

The interpolated mode uses two arbitrary motion vectors to predict from the two reference (anchor) frames. Both the direct and interpolated motion modes use round-up averaging for combining the two references into one reference macro block used in prediction. The interpolation process is exactly as is explained in the P frame section.

Direct mode is a very useful and bit-efficient mode in B-frames. By using this mode, we effectively do temporal interpolation and do not send any additional motion vectors, which are implicitly derived from the corresponding MB of the future P frame (Figure 66). The direct mode also frequently becomes associated with skipped MB's, especially at lower bit rates, thus becoming very powerful in controlling bit rate in the V9 codec.
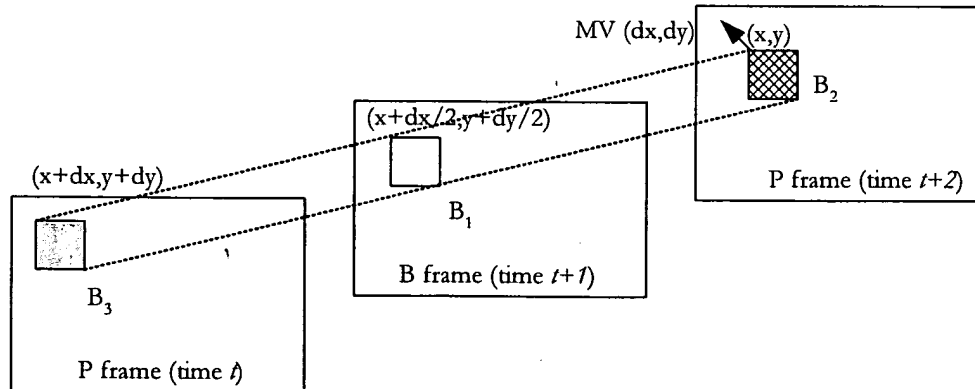
Figure 66: Direct Mode Prediction

In our implementation we have additionally de-coupled the strict temporal identity (or timestamp) of each individual B-frame and the factors used for the temporal interpolation, thus giving an additional dimension of optimization for the encoder. For example, when 2 B frames are used, in the usual interpretation of the direct mode, the first of 2 B frames uses the factor 1/3 to scale motion vectors from the forward reference and 2/3 to scale motion vectors from the backward reference picture. However, we send an independent scale fraction, thereby allowing us to associate arbitrary scale factors with each individual B-frame. Furthermore, we are able to switch both the effective inter-P distance as well as the number of B frames used in a GOP dynamically by means of this technique.

This effectively relaxes the constant velocity approximation of the traditional direct mode (e.g. MPEG, H.26x) to one of variable acceleration, and we are able to use it more effectively in a larger variety of content.

4.6.3.3 Motion Vector Prediction

The forward and backward components of the motion vectors are predicted independently. Macro blocks that use the direct or interpolate modes have valid forward and backward motion vectors associated with them. Macro blocks that are coded as forward or backward do not have valid backward and forward components respectively. For these cases, the direct mode motion vectors used in backward and forward directions respectively are used to fill in.

For intra coded macro blocks, the "intra motion vector" is used to fill in both forward and backward motion prediction planes.

4.6.3.4 Motion Vector Transmission

Whether or not a macro block is coded as direct is known at the start of decoding macro block level information. Non-direct macro blocks have one or two associated *residual* motion vectors, direct macro blocks have none. Skipped macro blocks that are direct coded have zero residual DCT coefficients (DCT AC for intra macro blocks), and skipped non-direct coded macro blocks have zero residual motion as well.

It is possible to gain some efficiency by coding the mode of the non-direct macro block *after* sending the first motion vector. Since WMV9 jointly codes motion vector information with the intra flag, intra macro blocks are identified after decoding the first motion vector. It is not necessary to send any mode information subsequently after an intra motion vector is received.

When the first motion vector is non-intra, the macro block type is sent. This is based on the efficient remapping of forward and backward into short and long types as explained earlier. The second motion vector is sent only if the macro block is interpolated, and if the iLast component of the first motion vector is nonzero. If the iLast component is zero for an interpolated macro block, it is implied that the second residual motion vector of the interpolated block is zero, and so are the residual DCT terms.

4.6.3.5 Subpixel Interpolation

Subpixel interpolation of B frames is performed in the same manner as interpolation of P frames. The valid modes are quarter pel bicubic and half pel bilinear.

4.6.3.6 Pixel Averaging

Macroblocks that are coded using the direct or interpolated modes have two associated predictions drawn from the two reference anchors. These predictions are merged into one by averaging. A pixelwise mean operation with upwards rounding is employed to perform averaging.

4.6.3.7 Reconstructing and Adding Error

The decoding, dequantization, IDCT, error addition and clamping of residuals to the predicted macro blocks is performed in a manner identical to that used in P frames. Intra macro blocks are also coded as they would be in P frames. However, the overlapped smoothing operation applied to edges between intra blocks in P frames is not performed in B frames.

4.7 Interlace B Frame Decoding

4.7.1 Picture Layer Decoding

In a field-coded macroblock, two motion vectors are associated with the macroblock if the B frame mode is either forward or backward (no motion vectors are sent for the direct mode). If the mode for the macroblock is interpolated, then four motion vectors may be sent, two each for the forward and backward reference frames. In frame mode, one motion vector is associated with the macroblock in forward and backward mode, zero in the direct mode and two in the interpolated mode.

The picture layer decoding of interlaced B frames is identical to Progressive B frame picture level decoding with the following modifications (see Figure 12 and Figure 13). The flag (INTERLCF) is sent in place of MVMODE and the bit plane code for field/frame mode, INTERLCMB is sent immediately following it.

4.7.2 Macroblock Layer Decoding

Macroblocks in interlaced B pictures can be one of 12 possible types: (forward, backward, direct, interpolated) x (Frame-code, Field-coded and Skipped). The macroblock type is indicated by a combination of picture and macroblock layer fields.

4.7.2.1   Frame-coded Macroblocks

Figure Figure 20 describes frame-coded B-macroblock decoding. Unless bit-plane coded at the picture level, DIRECTBIT is sent followed by BMVTYPE which indicates the type of B macroblock. This is optionally followed by the skip bit for the macroblock unless this was bit plane coded at the picture level. This is followed by zero, one or two motion vectors MVB1 and MVB2 depending on whether BMVTYPE is direct, forward/backward or interpolated. Following this, i.e. CBPCY onwards the syntax is identical as Inter Frame MB Layer decoding.

4.7.2.2   Field-coded Macroblocks

Figure 22 describes field-coded B-macroblock decoding. Unless bit-plane coded at the picture level, DIRECTBIT is sent followed by BMVTYPE which indicates the type of B macroblock. This is optionally followed by the SKIPBIT for the macroblock unless this was bit plane coded at the picture level. The CBPCY field that indicates which fields have field MVDATA fields present in the bitstream. TOPMVDATA field is associated with the top field blocks (i.e. blocks Y_0, Y_1, U_0, V_0) and a second BOTMVDATA field is associated with the bottom field blocks (i.e. blocks Y_2, Y_3, U_1, V_1).This is followed by zero, one or two motion vectors TOPMVB1 and TOPMVB2 and zero, one or two motion vectors BOTMVB1 and BOTMVB2 depending on whether BMVTYPE is direct, forward/backward or interpolated. Following this, i.e. SUBBLKPATU onwards the syntax is identical as Inter Field MB Layer decoding.

Motion vector prediction and computation, interpolation in the frame and field modes, in-loop deblocking etc. for interlaced B- frames are exactly the same as interlaced P- frames.

4.8   Overlapped Transform

If the sequence layer field OVERLAP is set to 1, then a filtering operation is conditionally performed across edges of two neighboring Intra blocks, for both the luminance and chrominance channels. This filtering operation (referred to as *overlap smoothing*) is performed subsequent to decoding the frame, and prior to in-loop deblocking. However, in the reference software, overlap smoothing is done after the relevant macroblock slices are decoded – this is functionally equivalent to smoothing after decoding the entire frame.

Overlapped transforms are modified block based transforms that exchange information across the block boundary. With a well designed overlapped transform, blocking artifacts can be minimized. For intra blocks, WMV9 simulates an overlapped transform by coupling an 8x8 DCT-like block transform with overlap smoothing. Edges of an 8x8 block that separate two intra blocks are smoothed – in effect an overlapped transform is implemented at this interface.

Figure 67 shows a portion of a P frame with I blocks. This could be either the Y or U/V channel. I blocks are gray (or crosshatched) and P blocks are white. The edge interface over which overlap smoothing is applied is marked with a crosshatch pattern. Overlap smoothing is applied to two pixels on either side of the separating boundary. The right bottom area of frame is shown here as an example. Pixels occupy individual cells and blocks are separated by heavy lines. The dark circle marks the 2x2 pixel corner subblock that is filtered in both directions.

The lower inset in Figure 67 shows four labeled pixels, a0 and a1 are to the left and b1, b0 to the right of the vertical block edge. The upper inset shows pixels marked p0, p1, q1 and q0 straddling a horizontal edge. The next section describes the filter applied to these four pixel locations.
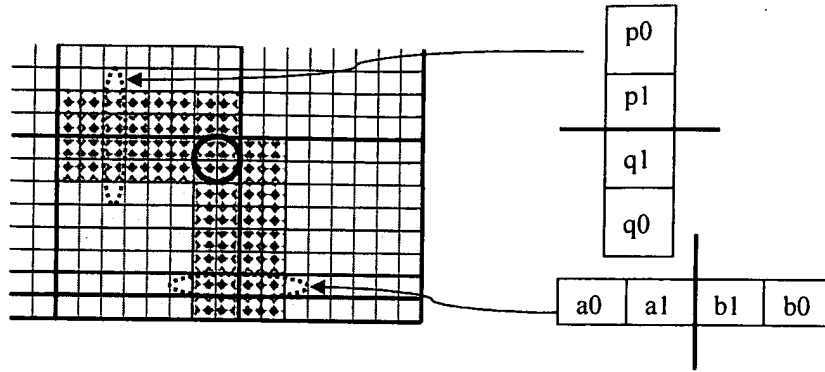
Figure 67: Example showing overlap smoothing

4.8.1 Overlap Smoothing

Overlap smoothing is applied subject to the following conditions:

1. All 8x8 block boundaries (except those at the periphery of the frame) are smoothed for I frames

2. Only block boundaries separating two intra blocks are smoothed for P frames

3. No overlap smoothing is performed for B frames

4. Subject to the above, overlap smoothing is applied only if the frame level quantization step size PQUANT is 9 or above

5. There is no dependence on DQUANT or differential quantization across macroblocks

Overlap smoothing is carried out on the unclamped 16 bit reconstruction. This is necessary because the forward process associated with overlap smoothing may result in range expansion beyond the permissible 8 bit range for pixel values. The result of overlap smoothing is clamped down to 8 bits, in line with the remainder of the pixels not touched by overlap smoothing.

Vertical edges (pixels a0, a1, b1, b0 in the above example) are filtered first, followed by the horizontal edges (pixels p0, p1, q1, q0). The intermediate result following the first stage of filtering (vertical edge smoothing) is stored in 16 bit. The core filters applied to the four pixels straddling either edge are given below:

$$
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \left(\begin{pmatrix} 7 & 0 & 0 & 1 \\ -1 & 7 & 1 & 1 \\ 1 & 1 & 7 & -1 \\ 1 & 0 & 0 & 7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} r_0 \\ r_1 \\ r_0 \\ r_1 \end{pmatrix} \right) >> 3
$$

The original pixels being filtered are (x0, x1, x2, x3). r0 and r1 are rounding parameters, which take on alternating values of 3 and 4 to ensure statistically unbiased rounding. The original values are filtered by the matrix with entries that are clearly easy to implement. These values, after adding the rounding factors, are bit shifted by three bits to give the filtered output (y0, y1, y2, y3).

For both horizontal and vertical edge filters, the rounding values are r0 = 4, r1 = 3 for odd-indexed columns and rows respectively, assuming the numbering within a block to start at 1. For even-indexed columns / rows, r0 = 3 and r1 = 4. Filtering is defined as an in-place 16 bit operation − thus the original pixels are overwritten after smoothing. For vertical edge filtering, the pixels (a0, a1, b1, b0) correspond to (x0, x1, x2, x3), which in turn get filtered to (y0, y1, y2, y3). Likewise, for horizontal edge filtering, the correspondence is with (p0, p1, q1, q0) respectively.

Pixels in the 2x2 corner, shown by the dark circle in Fig. XXX, are filtered in both directions. The order of filtering determines their final values, and therefore it is important to maintain the order − vertical edge filtering followed by horizontal edge filtering − for bit exactness. Conceptually, clamping is to be performed subsequent to the two directional filtering stages, on all pixels that are filtered. However, there may be some computational advantage to combining clamping with filtering − this is an implementation issue as long as it is done carefully to generate the correct output.

In the reference software, OverlapMBRow( ) performs overlap smoothing across a row of macroblocks. This function, and its descendants describe the exact process of overlap smoothing.

4.9 In-loop Deblock Filtering

If the sequence layer field LOOPFILTER = 1 then a filtering operation is performed on each reconstructed frame. This filtering operation is performed prior to using the reconstructed frame as a reference for motion predictive coding. Therefore, it is necessary that the decoder perform the filtering operation strictly as defined.

Since the intent of loop filtering is to smooth out the discontinuities at block boundaries the filtering process operates on the pixels that border neighboring blocks. For P pictures, the block boundaries can occur at every 4th, 8th, 12th, etc pixel row or column depending on whether an 8x8, 8x4 or 4x8 IDCT is used. For I pictures filtering occurs at every 8th, 16th, 24th, etc pixel row and column.

4.9.1 I Picture In-loop Deblocking

For I pictures, deblock filtering is performed at all 8x8 block boundaries. Figure 68 and Figure 69 show the pixels that are filtered along the horizontal and vertical border regions. The figures show the upper left corner of a component (luma, Cr or Cb) plane. The crosses represent pixels and the circled crosses represent the pixels that are filtered.
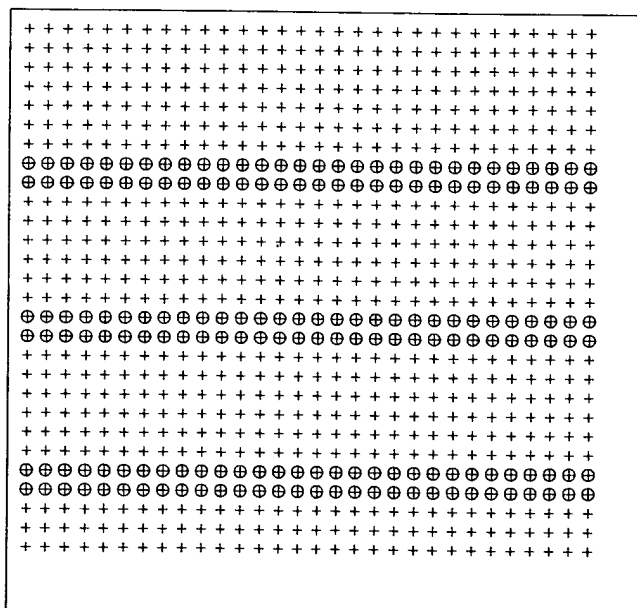
Figure 68: Filtered horizontal block boundary pixels in I picture
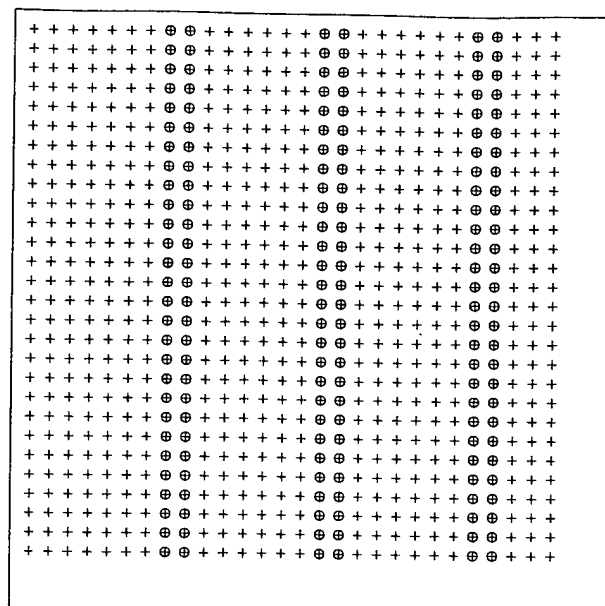
Figure 69: Filtered vertical block boundary pixels in I picture

As the figures show, the top horizontal line and first vertical line are not filtered. Although not depicted, the bottom horizontal line and last vertical line are also not filtered. In more formal terms, the following lines are filtered:

N = the number of horizontal 8x8 blocks in the plane (N*8 = horizontal frame size)

M = the number of vertical 8x8 blocks in the frame (M*8 = vertical frame size)

Horizontal lines (7,8), (15,16) ... ((N – 1)*8 – 1, (N –1)*8) are filtered

Vertical lines (7, 8), (15, 16) ... ((M-1)*8 - 1, (M – 1)*8) are filtered

The order in which the pixels are filtered is important. All the horizontal boundary lines in the frame are filtered first followed by the vertical boundary lines.

4.9.2 P Picture In-loop Deblocking

For P pictures, blocks may be Intra or Inter-coded. Intra-coded blocks always use an 8x8 IDCT to transform the DCT coefficients and the 8x8 block boundaries are always filtered. Inter-coded blocks may use an 8x8, 8x4, 4x8 or 4x4 IDCT to transform the DCT coefficients that represent the residual error. Depending on the status of the neighboring blocks, the boundary between the current and neighboring blocks may or may not be filtered. The decision of whether to filter a block or subblock border is as follows:

1) The boundaries between coded (at least one non-zero coefficient) subblocks (8x4, 4x8 or 4x4) within an 8x8 block are always filtered.

2) The boundary between a block or subblock and a neighboring block or subblock is not filtered if both have the same motion vector and both have no residual error (no DCT coefficients). Otherwise it is filtered.
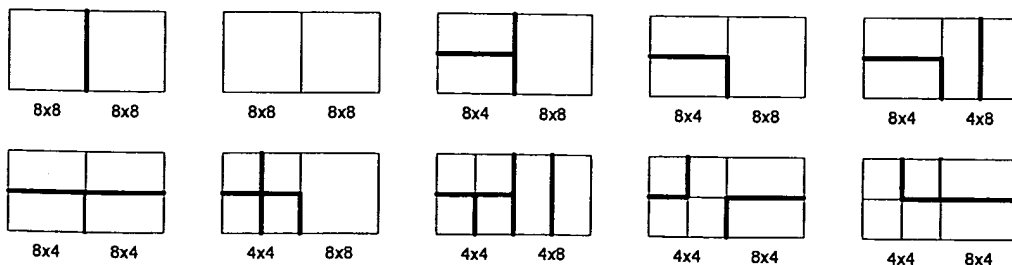
Figure 70 shows examples of when filtering between neighboring blocks does and does not occur. In this example it is assumed that the motion vectors for both blocks is the same (if the motion vectors are different then the boundary is always filtered). The shaded blocks or subblocks represent the cases where at least one nonzero coefficient is present. Clear blocks or subblocks represent cases where no DCT coefficients are present. Thick lines represent the boundaries that are filtered. Thin lines represent the boundaries that are not filtered. These examples illustrate only horizontal neighbors. The same applies for vertical neighbors.
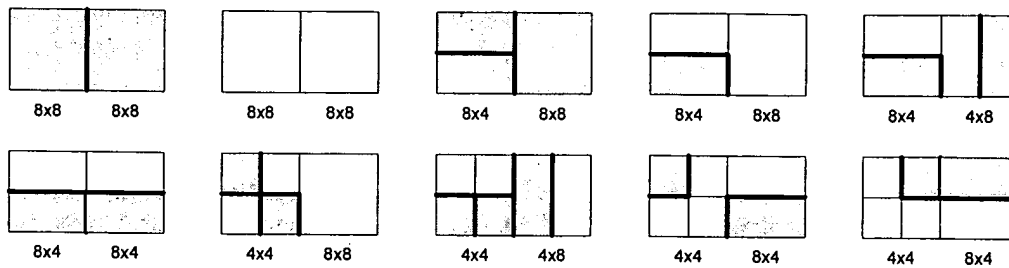
Figure 70: Example filtered block boundaries in P frames

Figure 71 and Figure 72 shows an example of the pixels that could be filtered in a P frame. The crosses represent pixel locations and the circled crosses represent the boundary pixels that will filtered if the conditions specified above are met.

Figure 71 shows pixels filtered along horizontal boundaries. As the figure shows, the pixels on either side of the block or subblock boundary are candidates to be filtered. For the horizontal boundaries this could be every 4th and 5th, 8th and 9th, 12th and 13th etc pixel row in the frame as these are the 8x8 and 8x4 horizontal boundaries.

Figure 72 shows pixels filtered along vertical boundaries. For the vertical boundaries, every 4th and 5th, 8th and 9th, 12th and 13th etc pixel column in the frame can be filtered as these are the 8x8 and 4x8 vertical boundaries.

The first and last row and the first and last column in the frame are not filtered.

The order in which pixels are filtered is important. First, all the 8x8 block horizontal boundary lines in the frame are filtered starting from the top line. Next, all 8x4 block horizontal boundary lines in the frame are filtered starting from the top line. Next, all 8x8 block vertical boundary lines are filtered starting from the leftmost line. Last, all 4x8 block vertical boundary lines are filtered starting with the leftmost line. In all cases, the rules specified above are used to determine whether the boundary pixels are filtered for each block or subblock.
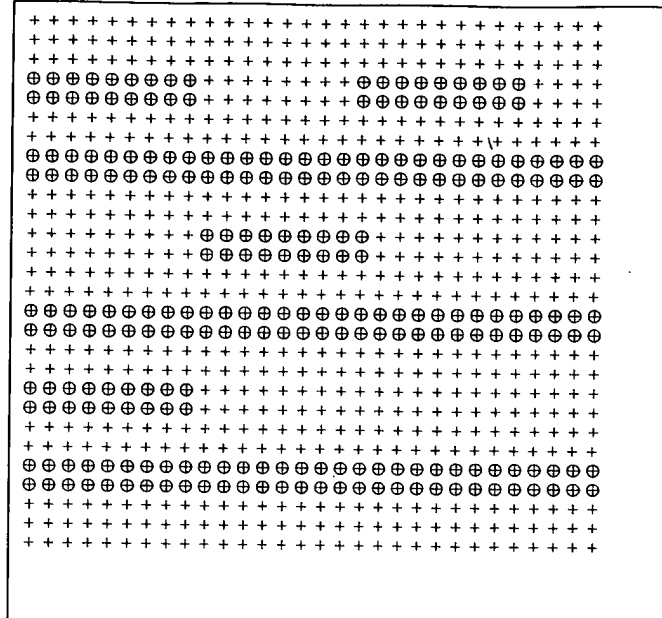
Figure 71: Horizontal block boundary pixels in P picture
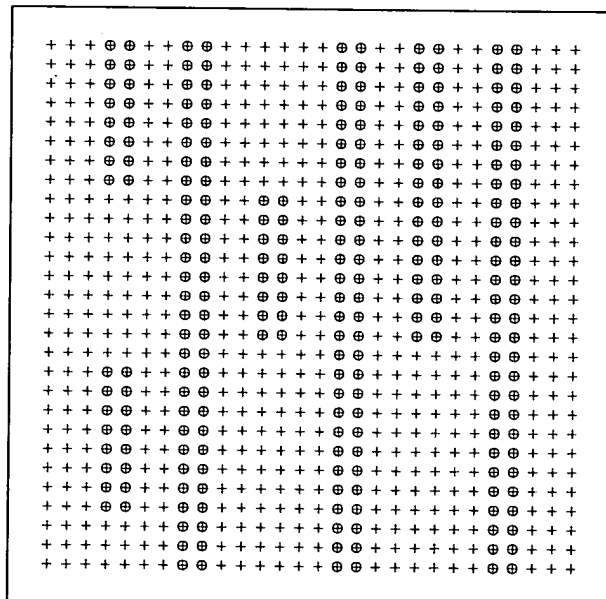
Figure 72: Vertical block boundary pixels in P picture

4.9.3   Interlace Picture In-loop Deblocking

If the sequence layer field LOOPFILTER = 1 then a filtering operation is performed on each reconstructed frame. This filtering operation is performed prior to using the reconstructed frame as a reference for motion predictive coding. Therefore, it is necessary that the decoder perform the filtering operation strictly as defined.

For I and P pictures filtering can occur for each pixels located immediately on the left and right of a vertical block boundary except for those located on the picture boundaries (i.e. the first and last column of the luminance and chrominance component). More specifically, the luminance block boundaries are located in between every $(k*8, y)^{th}$ and $(k*8 + 1, y)^{th}$ pixel where $k = 1, 2, ..., WidthY/8 - 1$, and $y = 1, ..., HeightY$. The chrominance block

boundaries are located in between every $(k*4, y)^{th}$ and $(k*4 + 1, y)^{th}$ pixel where $k = 1, 2, ...,$ WidthUV/4 $- 1,$ and $y = 1, ...,$ HeightUV.

We marked the pixels that are candidates for filtering in a typical MB in Figure 73 where M denote boundary pixels located across macroblock boundaries while B denote boundary pixels located within macroblock.

The decision on whether we should filter across a boundary is decided on a block by block basis. Remember that for a frame MB, a block contains 8 consecutive lines and for a field MB, a block contains either the top field (corresponding to lines 1, 3, 5, ..., 15 physically) or the bottom field (corresponding to lines 2, 4, 6, ..., 16) as described in section 2.2. This means that we are making the decision to filter 8 lines at a time.

The decision rule for deciding to filter across a block boundary is dependent on six pieces of information: The current block (CB)'s and the left neighboring block (LB)'s type (i.e. frame MB or field MB), whether it's intra or inter coded, and it's CBP (i.e. are there nonzero DCT coefficients). In general, the block boundary pixels are filter unless the following condition is met. If CB's type equals to LB's type and both block are not intra coded and both CBP's are zero, then we don't filter the block boundary. We do not make addition test for the chroma block boundaries. Instead, the chroma block boundaries are filtered if the corresponding luminance block boundaries are filtered (i.e. a one to one correspondence between the luma pixels (marked M in figure) and the chroma pixels (also marked M is figure). These rules apply to both I picture and P picture.

Once a block boundary is decided to be filtered we apply the V9 loop filter. Remember for frame block, we apply the filter to eight consecutive lines while for field block, we apply the filter to the 8 field lines.

M - Block boundary pixels across MB.
B - Block boundary pixels within MB.
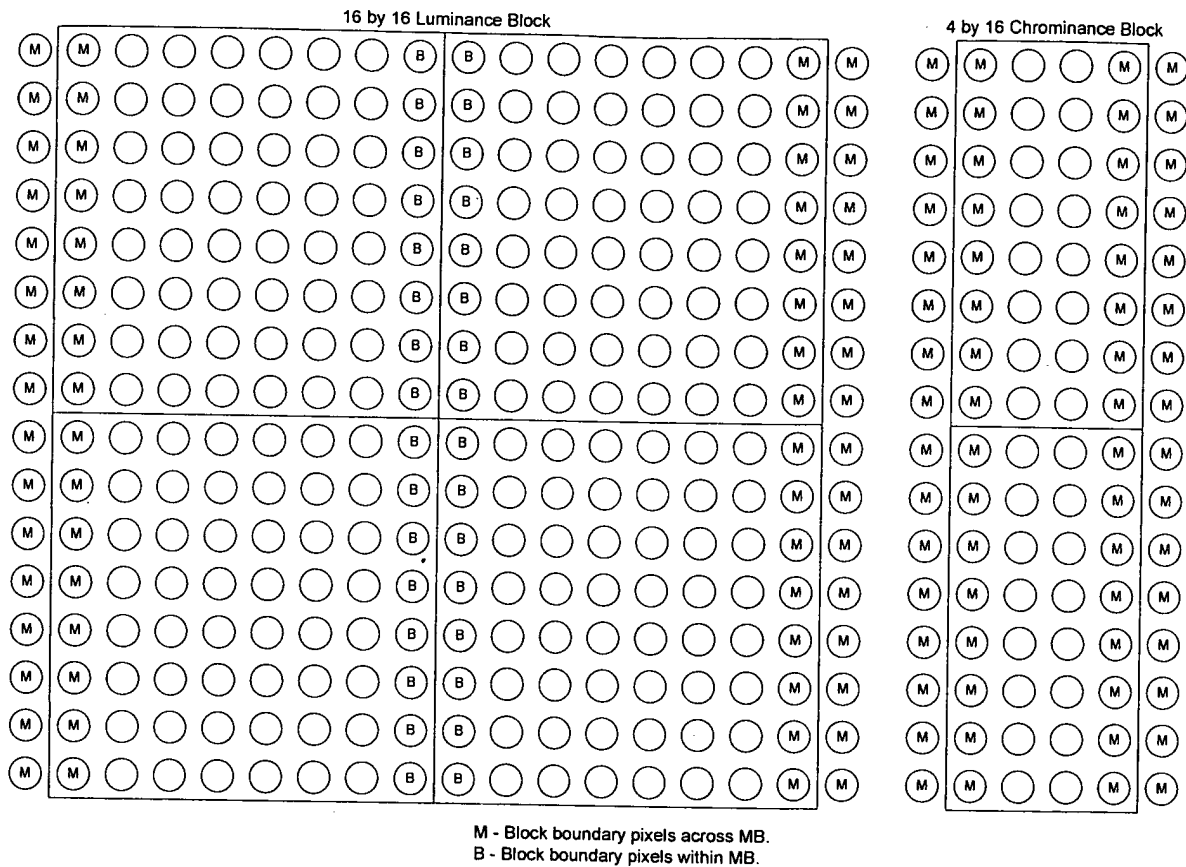
Figure 73: Filtered vertical block boundary pixels in a MB

4.9.4 Filter Operation

This section describes the filtering operation that is performed on the boundary pixels in I and P frames.

For P frames the decision criteria listed in section 4.9.2 determines which vertical and horizontal boundaries are filtered. For I frames, all the 8x8 vertical and horizontal boundaries are filtered. Since the minimum number of consecutive pixels that will be filtered in a row or column is four and the total number of pixels in a row or column will always be a multiple of four the filtering operation is performed on segments of four pixels.

For example, if the eight pixel pairs that make up the vertical boundary between two blocks is filtered then the eight pixels are divided into two 4-pixel segments as shown in Figure 74. In each 4-pixel segment, the third pixel pair is filtered first as indicated by the X's. The result of this filter operation determines whether the other three pixels in the segment are also filtered as described below.
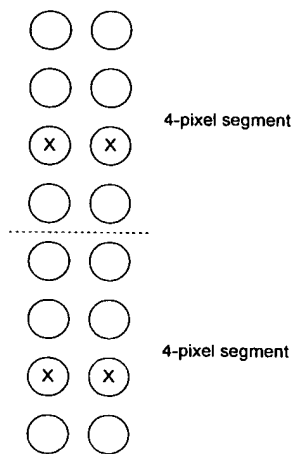
Figure 74: Four-pixel segments used in loop filtering

Figure 75 shows the pixels that are used in the filtering operation performed on the 3rd pixel pair. Pixels P4 and P5 are the pixel pairs that may be changed in the filter operation.
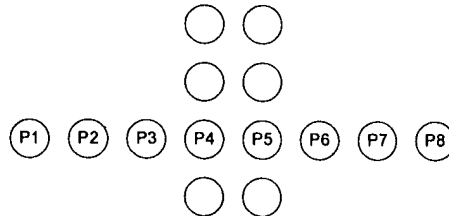
Figure 75: Pixels used in filtering operation

The pseudocode of Figure 76 shows the filtering operation performed on the 3rd pixel pair in each segment. The value filter_other_3_pixels indicates whether the remaining 3 pixel pairs in the segment are also filtered. If filter_other_3_pixels = true then the other three pixel pairs are filtered. If filter_other_3_pixels = false then they are not filtered and the filtering operation proceeds to the next 4-pixel segment. The pseudocode of Figure 77 shows the filtering operation that is performed on the 1st, 2nd and 4th pixel pairs if filter_other_3_pixels = true.

```
filter_other_3_pixels = true
a0 = (2*(P3 - P6) - 5*(P4 – P5) + 4) >> 3
if (|a0| < PQUANT) {
      a1 = (2*(P1 - P4) - 5*(P2 - P3) + 4) >> 3
      a2 = (2*(P5 - P8) - 5*(P6 - P7) + 4) >> 3
      a3 = min(|a1|, |a2|)
      if (a3 < |a0|)
      {
            d = 5*((sign(a0) * a3) - a0)/8
            clip = (P4 – P5)/2
            if (clip == 0)          /
                  filter_other_3_pixels = false
            else
            {
                  if (clip > 0)
                  {
```

```
                    if (d < 0)
                            d = 0
                    if (d > clip)
                            d = clip

            }
            else
            {
                    if (d > 0)
                            d = 0
                    if (d < clip)
                            d = clip

            }
            P4 = P4 - d
            P5 = P5 + d

        }
    }
    else
        filter_other_3_pixels = false

}
else

    filter_other_3_pixels = false
```

Figure 76: Pseudo-code illustrating filtering of 3rd pixel pair in segment

```
a0 = (2*(P3 - P6) - 5*(P4 – P5) + 4) >> 3
if (|a0| < PQUANT)
{
    a1 = (2*(P1 - P4) - 5*(P2 - P3) + 4) >> 3
    a2 = (2*(P5 - P8) - 5*(P6 - P7) + 4) >> 3
    a3 = min(|a1|, |a2|)
    if (a3 < |a0|)
    {
        d = 5*((sign(a0) * a3) - a0)/8
        clip = (P4 – P5)/2

        if (clip > 0)
        {
            if (d < 0)
                    d = 0
            if (d > clip)
                    d = clip
            P4 = P4 - d
            P5 = P5 + d
        }
        else if (clip < 0)
        {
            if (d > 0)
                    d = 0
            if (d < clip)
                    d = clip
```

```
                P4 = P4 - d
                P5 = P5 + d
            }
        }
    }
```

Figure 77: Pseudo-code illustrating filtering of 1ˢᵗ, 2ⁿᵈ and 4ᵗʰ pixel pair in segment

This section used the vertical boundary for example purposes. The same operation is used for filtering the horizontal boundary pixels.

4.10 Bitplane Coding

Certain macroblock-specific information can be encoded in one bit per macroblock. For example, whether or not any information is present for a macroblock (ie, whether or not it is skipped) can be signaled with one bit. In these cases, the status for all macroblocks in a frame can be coded as a bitplane and transmitted in the frame header. WMV9 uses bitplane coding in three cases to signal information about the macroblocks in a frame. These are: 1) signaling skipped macroblocks, 2) signaling field or frame macroblock mode and 3) signaling 1-MV or 4-MV motion vector mode for each macroblock. This section describes the bitplane coding scheme.

Frame-level bitplane coding is used to encode two-dimensional binary arrays. The size of each array is *rowMB* × *colMB*, where *rowMB* and *colMB* are the number of macroblock rows and columns respectively. Within the bitstream, each array is coded as a set of consecutive bits. One of seven modes is used to encode each array.

The seven modes are enumerated below.

1. *Raw mode* – coded as one bit per symbol

2. *Normal-2 mode* – two symbols coded jointly

3. *Differential-2 mode* – differential coding of bitplane, followed by coding two residual symbols jointly

4. *Normal-6 mode* – six symbols coded jointly

5. *Differential-6 mode* – differential coding of bitplane, followed by coding six residual symbols jointly

6. *Rowskip mode* – one bit skip to signal rows with no set bits

7. *Columnskip mode* – one bit skip to signal columns with no set bits

Section 3.3 shows the syntax elements that make up the bitplane coding scheme. The follow sections describe how to decode the bitstream and reconstruct the bitplane.

4.10.1 INVERT

The INVERT field shown in the syntax diagram of Figure 28 is a one bit code, which if set indicates that the bitplane has more set bits than zero bits. Depending on INVERT and the mode, the decoder must invert the interpreted bitplane to recreate the original.

4.10.2 IMODE

The IMODE field shown in the syntax diagram of Figure 28 encodes the mode used code the bitplane. The seven modes are described in section 4.10.3. Table 57 shows the codetable used to encode the IMODE field.

Table 57: IMODE Codetable

CODING MODE	CODEWORD
Raw	0000
Norm-2	10

Diff-2	001
Norm-6	11
Diff-6	0001
Rowskip	010
Colskip	011

4.10.3 DATABITS

The DATABITS field shown in the syntax diagram of Figure 28 is an entropy coded stream of symbols that is based on the coding mode. The seven coding modes are described in the following sections.

4.10.3.1 Description of Diff^1 Operation

Two of the coding modes (Diff-2 or Diff-6) employ an inverse differential coding method denoted $\textbf{Diff}^1$. If either differential mode is used, a bitplane of "differential bits" is first decoded using the corresponding normal modes (Norm-2 or Norm-6 respectively). The differential bits are used to regenerate the original bitplane. The regeneration process is a 2-D DPCM on a binary alphabet. In order to regenerate the bit at location (i, j), the predictor is generated according to

$$\hat{b}(i, j) = \begin{cases} \text{INVERT} & i = j = 0, \text{ or } b(i, j-1) \neq b(i-1, j) \\ b(0, j-1) & i == 0 \\ b(i-1, j) & \text{otherwise} \end{cases}$$

Bitwise inversion based on INVERT is not performed if differential coding is used.

4.10.3.2 Raw mode

In this mode, the bitplane is encoded as one bit per pixel scanned in the natural scan order. DATABITS is $rowMB \times colMB$ bits in length.

4.10.3.3 Normal-2 mode

If $rowMB \times colMB$ is odd, the first symbol is encoded raw. Subsequent symbols are encoded pairwise, in natural scan order. The binary VLC table in Table 58 is used to encode symbol pairs.

Table 58: Norm-2/Diff-2 Code Table

SYMBOL 2N	SYMBOL 2N + 1	CODEWORD
0	0	0
1	0	100
0	1	101
1	1	11

4.10.3.4 Diff-2 mode

The Normal-2 method is used to produce the bitplane as described in section 4.10.3.3 and then the $\textbf{Diff}^1$ operation is applied to the bitplane as described in section 4.10.3.9.

4.10.3.5 Normal-6 mode

In the *Norm-6* and *Diff-6* modes, the bitplane is encoded in groups of six pixels. These pixels are grouped into either 2x3 or 3x2 tiles. The bitplane is tiled maximally using a set of rules, and the remaining pixels are encoded using a variant of *row-skip* and *column-skip* modes.

3x2 "vertical" tiles are used if and only if $rowMB$ is a multiple of 3 and $colMB$ is not. Else, 2x3 "horizontal" tiles are used, as shown in Figure 78.
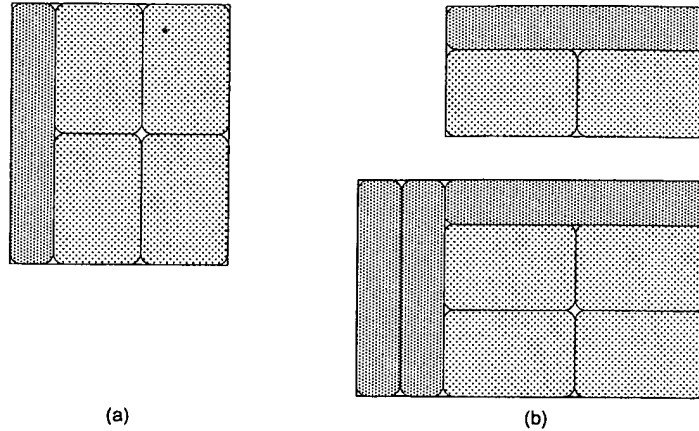
Figure 78: An example of 3x2 "vertical" tiles (a) and 2x3 "horizontal" tiles (b) – the elongated dark rectangles are 1 pixel wide and encoded using row-skip and column-skip coding.

The 6-element tiles are encoded first, followed by the *column-skip* and *row-skip* encoded linear tiles. If the array size is a multiple of 3x2 or of 2x3, the latter linear tiles do not exist and the bitplane is perfectly tiled.

The 6-element rectangular tiles are encoded using an incomplete Huffman code, i.e. a Huffman code which does not use all end nodes for encoding. Let N be the number of set bits in the tile, i.e. $0 \le N \le 6$. For $N < 3$, a VLC is used to encode the tile. For $N = 3$, a fixed length escape is followed by a 5 bit fixed length code, and for $N > 3$, a fixed length escape is followed by the code of the complement of the tile.

The rectangular tile contains 6 bits of information. Let k be the code associated with the tile, where $k = b_i \, 2^i$, b_i is the binary value of the ith bit in natural scan order within the tile. Hence $0 \le k \le 64$. Table 59 is used to encode k.

Table 59: Code table for 2x3 and 3x2 tiles

k	VLC / Escape symbol		Followed by	
	Codeword	Codelength	Codeword	Codelength
0	1	1		
1	2	4		
2	3	4		
3	0	8		
4	4	4		
5	1	8		
6	2	8		
7	2	5	3	5
8	5	4		
9	3	8		
10	4	8		
11	2	5	5	5
12	5	8		
13	2	5	6	5
14	2	5	7	5

15	3	5	14	8
16	6	4		
17	6	8		
18	7	8		
19	2	5	9	5
20	8	8		
21	2	5	10	5
22	2	5	11	5
23	3	5	13	8
24	9	8		
25	2	5	12	5
26	2	5	13	5
27	3	5	12	8
28	2	5	14	5
29	3	5	11	8
30	3	5	10	8
31	3	5	7	4
32	7	4		
33	10	8		
34	11	8		
35	2	5	17	5
36	12	8		
37	2	5	18	5
38	2	5	19	5
39	3	5	9	8
40	13	8		
41	2	5	20	5
42	2	5	21	5
43	3	5	8	8
44	2	5	22	5
45	3	5	7	8
46	3	5	6	8
47	3	5	6	4
48	14	8		
49	2	5	24	5
50	2	5	25	5
51	3	5	5	8
52	2	5	26	5
53	3	5	4	8

54	3	5	3	8
55	3	5	5	4
56	2	5	28	5
57	3	5	2	8
58	3	5	1	8
59	3	5	4	4
60	3	5	0	8
61	3	5	3	4
62	3	5	2	4
63	3	5	1	1

4.10.3.6 Diff-6 mode

The Normal-6 method is used to produce the bitplane as described in section 4.10.3.5 and then the **Diff^1** operation is applied to the bitplane as described in section 4.10.3.9.

4.10.3.7 Row-skip mode

In the row-skip coding mode, all-zero rows are skipped with one bit overhead. The syntax is as shown in Figure 79.
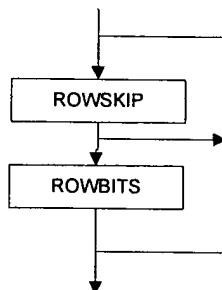
Figure 79: Syntax diagram of row-skip coding

If the entire row is zero, a zero bit is sent as the ROWSKIP symbol, and ROWBITS is skipped. If there is a set bit in the row, ROWSKIP is set to 1, and the entire row is sent raw (ROWBITS). Rows are scanned from the top to the bottom of the frame.

4.10.3.8 Column-skip mode

Column-skip is the transpose of row-skip. Columns are scanned from the left to the right of the frame.

4.10.3.9 Diff^1 : Inverse differential decoding

If either differential mode (Diff-2 or Diff-6) is used, a bitplane of "differential bits" is first decoded using the corresponding normal modes (Norm-2 or Norm-6 respectively). The differential bits are used to regenerate the original bitplane. The regeneration process is a 2-D DPCM on a binary alphabet. In order to regenerate the bit at location (i,j), the predictor is generated according to

$$\hat{b}(i,j) = \begin{cases} \text{INVERT} & i = j = 0, \text{ or } b(i,j-1) \neq b(i-1,j) \\ b(0,j-1) & i = 0 \\ b(i-1,j) & \text{otherwise} \end{cases}$$

132

Bitwise inversion based on INVERT is not performed if differential coding is used.

4.11 Start Codes

Start codes or sync markers are known sequences of bits that are inserted at important locations in the bitstream to clearly identify these locations. There are several reasons that require sync markers or start codes – the important ones are for error resilience and for parallel decoding of the bitstream.

Ideally, start codes must be unique and *non-emulatable* by the bitstream. In other words, bit patterns representing start codes must not occur in a normally coded bitstream. However, this places additional requirements on the bitstream syntax and/or on computations. WMV9 takes the simpler approach – start codes in WMV9 are not guaranteed to be unique. However, start codes are 24 bits in length and it is expected that even if they do randomly occur in a bitstream, such occurrences will be rare.

WMV9 places start codes only at byte boundaries. Assuming a uniform distribution, it can be expected that one start code will randomly be emulated in a 2^{24} byte long stream. For a bitrate of 1Mbps, this is equivalent to one random start code emulation every two minutes, or one occurrence every 3900 frames.

The sequence level flag STARTCODE determines whether start codes are enabled in the sequence. If they are enabled, start codes are sent only for I and P frames. No start codes are allowed in B frames, including B frames coded as Intra. When STARTCODE is enabled, all bitplanes are encoded as raw bitplanes and the relevant data (e.g. 4MV/1MV, skipbit) is sent at the macroblock level.

Start codes may only occur at the start of a row of macroblocks (also known as a *slice*). No start code is permitted in the first slice. When sequence level STARTCODE is enabled, a single bit is sent at the end of every slice, except for the last slice in the frame, to indicate whether or not a start code follows. If this bit is zero, it means that no start code follows. If this bit is 1, the remainder of the current byte is flushed out. Subsequently, the 24 bit byte aligned data is read from the bitstream. This is the start code.

Two start codes are defined in WMV9. These are the *short* and *long* start codes. Both the codes are 24 bits in length, but the *payload* or data following the start code differs in length. The short start code, whose hex representation is 0x000002, is followed by a 5 byte payload. The long start code (hex 0x000003) is followed by a 10 byte payload.

The windows media player does not handle the 5 or 10 byte payload in its current implementation. Currently, there is no requirement on the decoder to do anything with the payload in order to be compliant with the spec. The only interoperability requirement on any decoder is that the decoder correctly handle encoded content that may have embedded start codes, assuming that no errors are present in the bitstream.

Although there is some flexibility in the design of the payload of the start code, the following points must be noted:

1. WMV9 specifies that the payload only be used for the purpose of retransmitting relevant frame, slice or MB level information. WMV9 explicitly prohibits use of the slice code payload for sending information that is outside the purview of the codec (such as overlay text, hyperlinks etc.).

2. The slice code payload may be used to transmit the slice number – i.e. the index of the macroblock in the vertical direction.

3. The slice code payload may be used to transmit parity, error detection and error recovery information.

4. Part of the slice code payload may be used to implement a longer start code to reduce probability of emulation.

5. Frame level information may be divided up between slices to achieve redundancy. The longest possible frame header occupies less than 80 bits – the design choice for the payload was based on this fact.

6. WMV9 requires interoperability between decoders for non-corrupted bitstreams. Therefore, any implementation using slice codes must assume that the sequence should decode correctly when the slice code payload is disregarded, assuming of course that there are no errors in the bitstream.

7. Any error resilience introduced by good slice code payload design will work only with a decoder that implements the same error resilience mechanism – in effect being a walled garden extension.

Fig. XXX represents a coded (I or P) frame. Subfigure (a) shows successive macroblocks coded when STARTCODE is zero, (b) shows coded macroblocks when STARTCODE is one but no start codes are actually sent, and (c) shows the case when both long and short start codes are sent in the frame. The frame header, start codes and payloads are byte aligned. The trailing 0 or 1 in all but the last slice is sent when STARTCODE is 1.

This is necessary to ensure byte flushing in the case that a startcode is sent at the start of the next slice. "FB" in the figure stands for *flush byte* or the process of stuffing between zero and seven bits to reach the end of the current byte. "SC" and "PL" stand for start code and payload respectively. The start code 0x000002 (= 0x2) is followed by a 5 byte payload and 0x000003 (=0x3) is followed by a 10 byte payload.

There are no start codes in B frames. For B frames, the entropy coded stream follows the order shown in Fig. XXX (a) regardless of whether STARTCODE is 0 or 1.
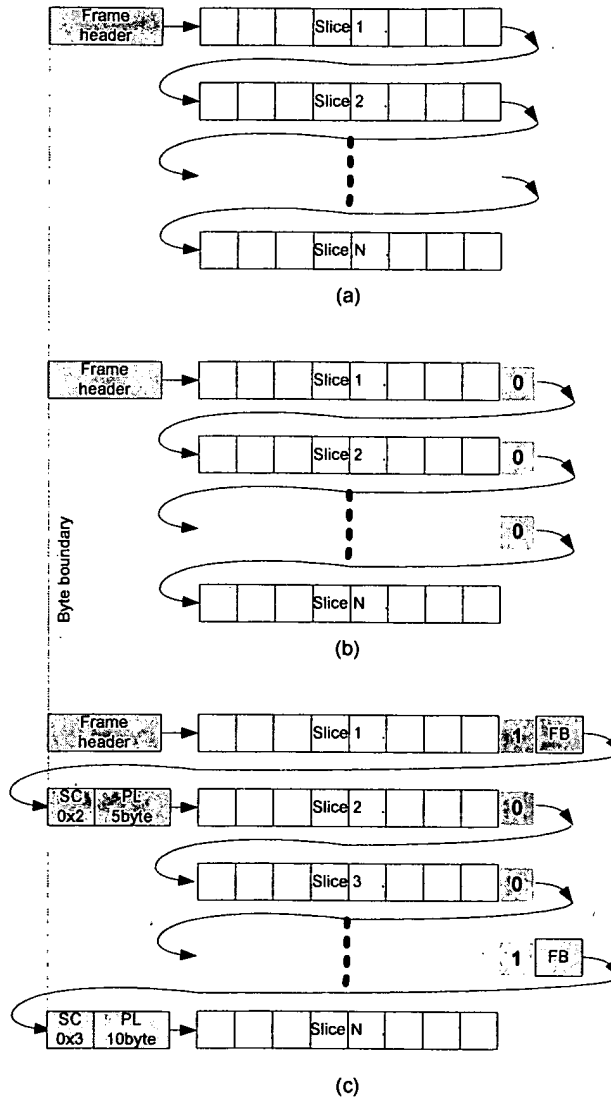
Fig. XXX : Start codes in WMV9 – (a) shows sequence of entropy coded data with STARTCODE set to zero, (b) STARTCODE is 1 but no start codes are actually sent and (c) STARTCODE is 1, a long and a short start code are sent, some slices do not have start codes

4.12 IDCT Conformance

The X8INTRA decoding process requires strict conformance with the IDCT implementation defined in Annex B. Also, due to the potential for rapid error accumulation in in-loop deblock filtering, it is highly recommended that strict compliance with IDCT defined in Annex B be used for baseline I frames and P frames.

4.13 WMV9 Inverse Transform Conformance

The decoding process requires strict conformance with the WMV9 Inverse Transform implementation defined in Annex C.

5 Tables

5.1 X8INTRA I-Picture Tables

! *4/3/2002: X8INTRA is disabled for the Main and Simple profiles in WMV9. It is described in this*
• *document only for completeness (since it is used in WMV8) and future use in the Complex profile of WMV9. The Complex profile of WMV9 is purely for technology demonstration, and is not intended for use by content providers and hardware partners.*

Table 60: Huffman code tables for DIFFORIENT, low rate

DIFFORIENT	Code set 0		Code set 1	
	Code	Length	Code	Length
0	0	2	0	1
1	1	2	2	2
2	4	3	6	3
3	5	3	1c	5
4	6	3	1d	5
5	38	6	78	7
6	1d	5	3d	6
7	39	6	79	7
8	3c	6	7c	7
9	3d	6	7d	7
10	3e	6	7e	7
11	3f	6	7f	7

Table 61: Huffman code tables for DIFFORIENT, high rate

DIFFORIENT	Code set 0		Code set 1		Code set 2		Code set 3	
	Code	Length	Code	Length	Code	Length	Code	Length
0	0	1	0	5	0	2	0	3
1	4	3	1	5	1	2	2	4
2	5	3	2	5	4	3	3	4
3	c	4	1	2	5	3	1	2
4	d	4	2	2	6	3	2	2
5	38	6	2	4	38	6	18	5
6	1d	5	3	5	39	6	19	5
7	39	6	6	3	1d	5	d	4

8	3c	6	3	4	3c	6	1c	5
9	3d	6	e	4	3d	6	1d	5
10	3e	6	1e	5	3e	6	1e	5
11	3f	6	1f	5	3f	6	1f	5

Table 62: Low rate code tables for *DCVALUE*

index	Code set 0		Code set 1		Code set 2		Code set 3		Code set 4		Code set 5		Code set 6		Code set 7	
	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length
0	0	5	0	3	0	7	0	4	0	8	0	7	0	5	0	7
1	1	4	1	3	1	7	1	3	1	7	1	7	1	4	1	7
2	2	4	2	3	1	6	2	3	2	7	1	6	1	5	2	7
3	1	5	6	4	4	7	1	4	3	7	2	6	4	5	3	7
4	6	5	7	4	3	6	6	4	2	9	3	6	a	6	4	7
5	4	4	4	3	5	7	4	3	8	8	4	6	6	5	5	7
6	7	5	a	4	10	8	5	3	3	9	5	6	b	6	6	7
7	a	5	b	4	11	8	6	3	240	14	6	6	e	6	7	7
8	2c	7	30	6	240	13	e	5	241	14	7	6	3c	8	8	7
9	2d	7	62	7	241	13	f	5	242	14	8	6	3d	8	9	7
10	5c0	12	63	7	242	13	70	7	243	14	9	6	7c0	13	a	7
11	5c1	12	640	11	243	13	710	11	244	14	a	6	7c1	13	b	7
12	5c2	12	641	11	244	13	711	11	245	14	b	6	7c2	13	c	7
13	5c3	12	642	11	245	13	712	11	246	14	c	6	7c3	13	d	7
14	5c4	12	643	11	246	13	713	11	247	14	d	6	7c4	13	e	7
15	5c5	12	644	11	247	13	714	11	124	13	e	6	7c5	13	f	7
16	5c6	12	645	11	124	12	715	11	125	13	f	6	7c6	13	10	7
17	3	3	33	6	1	1	1d	5	1	2	10	6	1	2	1	1
18	2	2	d	4	1	2	72	7	1	1	11	6	2	2	1	2
19	6	3	1c	5	1	3	3c	6	1	3	12	6	6	3	11	7
20	e	4	1d	5	3	5	3d	6	1	4	13	6	e	4	12	7
21	1e	5	3c	6	5	6	73	7	3	6	14	6	1e	5	13	7
22	1f	5	1f	5	13	8	7c	7	5	7	15	6	1f	5	14	7
23	2f	7	65	7	125	12	7d	7	13	9	16	6	3f	8	15	7
24	5d	8	7a	7	126	12	7e	7	126	13	17	6	7d	9	16	7
25	.5c7	12	646	11	127	12	716	11	127	13	18	6	7c7	13	17	7
26	5c8	12	7b	7	128	12	717	11	128	13	19	6	7c8	13	18	7
27	5c9	12	647	11	129	12	718	11	129	13	1	1	7c9	13	19	7
28	5ca	12	648	11	12a	12	7f	7	12a	13	1a	6	7ca	13	1a	7
29	5cb	12	649	11	12b	12	719	11	12b	13	1b	6	7cb	13	1b	7
30	5cc	12	64a	11	12c	12	71a	11	12c	13	1c	6	7cc	13	1c	7
31	5cd	12	64b	11	12d	12	71b	11	12d	13	1d	6	7cd	13	1d	7

| 32 | 5ce | 12 | 326 | 10 | 12e | 12 | 38e | 10 | 12e | 13 | 1e | 6 | 7ce | 13 | 1e | 7 |
| 33 | 5cf | 12 | 327 | 10 | 12f | 12 | 38f | 10 | 12f | 13 | 1f | 6 | 7cf | 13 | 1f | 7 |

Table 63: High rate code tables for *DCVALUE*

index	Code set 0		Code set 1		Code set 2		Code set 3		Code set 4		Code set 5		Code set 6		Code set 7	
	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length	Code	Length
0	0	5	0	4	0	4	0	4	0	7	0	5	0	4	0	6
1	1	4	1	3	1	3	1	2	1	7	1	4	1	2	1	6
2	1	5	2	3	2	3	1	3	2	7	1	5	1	3	2	6
3	4	5	1	4	1	4	4	3	3	7	8	6	4	3	6	7
4	5	5	6	4	6	4	5	3	4	7	9	6	1	4	7	7
5	6	5	4	3	7	4	6	3	5	7	a	6	a	4	4	6
6	e	6	7	4	8	4	1	4	6	7	16	7	16	5	5	6
7	f	6	5	3	9	4	e	4	7	7	c	6	2e	6	6	6
8	40	8	c	4	28	6	3c	6	8	7	17	7	5e	7	e	7
9	41	8	d	4	29	6	3d	6	9	7	d	6	5f	7	1e	8
10	840	13	1c	5	54	7	7c	7	a	7	38	8	c0	8	1f	8
11	841	13	3a	6	55	7	fa	8	b	7	1d	7	3040	14	40	9
12	842	13	1d8	9	ac0	12	3ec0	14	c	7	39	8	3041	14	82	10
13	843	13	1d9	9	ac1	12	3ec1	14	d	7	780	13	305	10	830	14
14	844	13	1da0	13	ac2	12	3ec2	14	e	7	781	13	183	9	831	14
15	845	13	1da1	13	ac3	12	3ec3	14	f	7	782	13	3042	14	832	14
16	846	13	1da2	13	ac4	12	1f62	13	10	7	783	13	3043	14	833	14
17	2	2	3c	6	b	4	1f7	9	1	1	2	3	d	4	1	1
18	3	2	3d	6	6	3	7e	7	1	2	1	1	7	3	1	2
19	3	3	3e	6	e	4	fe	8	11	7	3	3	19	5	3	4
20	5	4	77	7	1e	5	ff	8	12	7	1f	7	31	6	5	5
21	9	5	1db	9	3e	6	1f63	13	13	7	3d	8	c2	8	9	6
22	11	6	7e	7	3f	6	1f64	13	14	7	79	9	c3	8	11	7
23	43	8	fe	8	57	7	1f65	13	15	7	784	13	3044	14	21	8
24	85	9	1fe	9	ad	8	1f66	13	16	7	785	13	3045	14	834	14
25	847	13	1da3	13	ac5	12	1f67	13	17	7	786	13	3046	14	835	14
26	848	13	1da4	13	ac6	12	1f68	13	18	7	787	13	3047	14	836	14
27	849	13	1da5	13	ac7	12	1f69	13	19	7	788	13	3048	14	837	14
28	84a	13	ed3	12	ac8	12	1f6a	13	1a	7	789	13	3049	14	838	14
29	84b	13	ed4	12	ac9	12	1f6b	13	1b	7	78a	13	304a	14	839	14
30	84c	13	1ff	9	aca	12	1f6c	13	1c	7	78b	13	304b	14	83a	14
31	84d	13	ed5	12	acb	12	1f6d	13	1d	7	78c	13	304c	14	83b	14
32	84e	13	ed6	12	566	11	1f6e	13	1e	7	78d	13	304d	14	41e	13
33	84f	13	ed7	12	567	11	1f6f	13	1f	7	3c7	12	1827	13	41f	13

Table 64: Inter low rate code tables (S_INTER) for *ACVALUE*

index	code set 0 word	length	code set 1 word	length	code set 2 word	length	code set 3 word	length	code set 4 word	length	code set 5 word	length	code set 6 word	length	code set 7 word	length
0	0	3	0	3	0	4	0	8	0	9	0	10	0	2	0	3
1	2	4	2	4	2	5	1	8	1	9	1	10	4	4	2	4
2	c	6	3	4	180	12	2	8	2	9	2	10	a	5	6	5
3	d	6	8	5	181	12	3	8	3	9	3	10	b	5	7	5
4	1c	7	12	6	182	12	4	8	4	9	4	10	18	6	10	6
5	f	6	13	6	183	12	5	8	5	9	5	10	19	6	44	8
6	1d00	15	14	6	184	12	6	8	6	9	6	10	34	7	23	7
7	3b	8	15	6	185	12	7	8	7	9	7	10	6a	8	12	6
8	1d01	15	2c	7	186	12	8	8	8	9	8	10	6b	8	26	7
9	75	9	5a	8	187	12	9	8	9	9	9	10	6c	8	8a0	13
10	1d02	15	5b	8	188	12	a	8	a	9	a	10	da	9	4e	8
11	80	9	5c	8	189	12	b	8	b	9	b	10	db	9	4f	8
12	1d03	15	5d	8	c5	11	c	8	c	9	c	10	1b8	10	8a1	13
13	1d04	15	1780	14	c6	11	d	8	d	9	d	10	dd	9	8a2	13
14	1d05	15	179	10	c7	11	e	8	e	9	e	10	1b9	10	8a3	13
15	e83	14	17a	10	c8	11	f	8	f	9	f	10	3780	15	50	8
16	9	5	6	4	c9	11	10	8	10	9	10	10	4	3	6	4
17	11	6	e	5	ca	11	11	8	11	9	11	10	e	5	b	5
18	81	9	1e	6	cb	11	12	8	12	9	12	10	1e	6	29	7
19	82	9	3e	7	cc	11	13	8	13	9	13	10	1f	6	15	6
20	21	7	10	5	cd	11	14	8	14	9	a	9	a	4	1c	6
21	28	7	22	6	ce	11	15	8	15	9	b	9	58	7	3a	7
22	83	9	12	5	cf	11	16	8	b	8	c	9	17	5	1e	6
23	2	2	a	4	1	1	1	1	1	2	1	1	18	5	4	3
24	3	3	13	5	1	2	17	8	1	1	1	3	59	7	14	5
25	c	4	16	5	4	5	c	7	c	8	d	9	5a	7	15	5
26	d	4	23	6	5	5	d	7	d	8	e	9	5b	7	b	4
27	b	5	2e	6	6	5	e	7	e	8	1	2	c8	8	1f	6
28	15	6	2f	6	d0	11	f	7	f	8	f	9	65	7	30	6
29	52	8	30	6	d1	11	10	7	10	8	10	9	66	7	31	6
30	70	7	31	6	d2	11	11	7	11	8	11	9	c9	8	19	5
31	39	6	3f	7	d3	11	12	7	12	8	12	9	ce	8	51	8
32	71	7	5f	8	d4	11	13	7	13	8	13	9	cf	8	34	6
33	53	8	c8	8	d5	11	14	7	14	8	14	9	d0	8	35	6
34	e84	14	65	7	d6	11	15	7	15	8	15	9	d1	8	36	6
35	74	7	66	7	d7	11	16	7	16	8	16	9	d2	8	37	6
36	75	7	67	7	d8	11	17	7	17	8	17	9	d3	8	76	8
37	76	7	68	7	d9	11	18	7	18	8	18	9	df	9	77	8

	code		code		code		code		code		code		code		code	
38	1dc	9	c9	8	da	11	19	7	19	8	19	9	d4	8	70	7
39	1e	5	69	7	7	5	1a	7	1a	8	1a	9	d5	8	1d	5
40	3e	6	6a	7	db	11	1b	7	1b	8	1b	9	d6	8	71	7
41	1dd	9	d6	8	dc	11	1c	7	1c	8	1c	9	1ae	9	72	7
42	ef	8	d7	8	dd	11	1d	7	1d	8	1d	9	3781	15	8a4	13
43	1f8	9	d8	8	de	11	1e	7	1e	8	1e	9	1bd	10	73	7
44	1f9	9	1781	14	df	11	1f	7	1f	8	1f	9	35e	10	f0	8
45	e85	14	17b	10	e0	11	20	7	20	8	20	9	35f	10	8a5	13
46	e86	14	1b2	9	e1	11	21	7	21	8	21	9	3782	15	8a6	13
47	e87	14	1782	14	e2	11	22	7	22	8	22	9	360	10	8a7	13
48	fd	8	1c	5	e3	11	23	7	23	8	23	9	37	6	79	7
49	e88	14	1b3	9	e4	11	24	7	24	8	24	9	1b1	9	7a	7
50	e89	14	1783	14	e5	11	25	7	25	8	25	9	3783	15	8a8	13
51	e8a	14	1784	14	e6	11	26	7	26	8	26	9	3784	15	8a9	13
52	e8b	14	1d	5	e7	11	27	7	27	8	27	9	e	4	f1	8
53	e8c	14	da	8	e8	11	28	7	28	8	28	9	3c	6	8aa	13
54	e8d	14	1785	14	e9	11	29	7	29	8	29	9	361	10	8ab	13
55	e8e	14	1786	14	ea	11	2a	7	2a	8	2a	9	3785	15	8ac	13
56	e8f	14	1787	14	eb	11	2b	7	2b	8	2b	9	1bc3	14	8ad	13
57	e90	14	37	6	ec	11	2c	7	2c	8	2c	9	3d	6	f6	8
58	e91	14	db	8	ed	11	2d	7	2d	8	2d	9	d9	8	8ae	13
59	1fc	9	78	7	ee	11	2e	7	2e	8	2e	9	1bc4	14	7c	7
60	e92	14	f2	8	ef	11	2f	7	2f	8	2f	9	368	10	f7	8
61	e93	14	1e6	9	f0	11	30	7	30	8	30	9	1bc5	14	8af	13
62	e94	14	f4	8	f1	11	31	7	31	8	31	9	1bc6	14	8b0	13
63	e95	14	1788	14	f2	11	32	7	32	8	32	9	1bc7	14	8b1	13
64	e96	14	1789	14	f3	11	33	7	33	8	33	9	1bc8	14	8b2	13
65	e97	14	f5	8	f4	11	34	7 ·	34	8	34	9	db	8	fa	8
66	1fd	9	1e7	9	f5	11	35	7	35	8	35	9	369	10	8b3	13
67	e98	14	178a	14	f6	11	36	7	36	8	36	9	36a	10	8b4	13
68	1fe	9	178b	14	f7	11	37	7	37	8	37	9	1bc9	14	8b5	13
69	e99	14	178c	14	f8	11	38	7	38	8	38	9	1bca	14	8b6	13
70	e9a	14	178d	14	f9	11	39	7	39	8	39	9	1bcb	14	8b7	13
71	e9b	14	1ec	9	fa	11	3a	7	3a	8	3a	9	1bcc	14	fb	8
72	e9c	14	178e	14	fb	11	3b	7	3b	8	3b	9	1bcd	14	45c	12
73	1ff	9	1f	5	fc	11	3c	7	3c	8	3c	9	1f	5	3f	6
74	e9d	14	f7	8	fd	11	3d	7	3d	8	3d	9	36b	10	45d	12
75	e9e	14	1ed	9	fe	11	3e	7	3e	8	3e	9	1bce	14	45e	12
76	e9f	14	178f	14	ff	11	3f	7	3f	8	3f	9	1bcf	14	45f	12

Table 65: Inter high rate code tables (S_INTER) for *ACVALUE*

	code	set 0	code	set 1	code	set 2	code	set 3	code	set 4	code	set 5	code	set 6	code	set 7

index	word	length	word	length	word	length	word	length	word	length	word	length	word	length	word	length
0	0	2	0	3	0	2	0	2	0	3	0	2	0	2	0	4
1	2	3	4	5	2	3	2	3	2	4	4	4	4	4	80	11
2	6	4	14	7	3	3	6	4	3	4	a	5	a	5	81	11
3	e	5	b	6	8	4	e	5	8	5	b	5	16	6	82	11
4	1e	6	c	6	12	5	f	5	12	6	18	6	17	6	83	11
5	3e	7	2a	8	13	5	20	6	13	6	19	6	60	8	84	11
6	3f	7	2b	8	28	6	21	6	14	6	34	7	c2	9	85	11
7	40	7	34	8	29	6	44	7	2a	7	6a	8	186	10	86	11
8	104	9	d40	14	54	7	45	7	16	6	6b	8	187	10	87	11
9	83	8	d41	14	55	7	8c	8	2b	7	6c	8	c4	9	88	11
10	84	8	1b	7	56	7	8d	8	5c	8	da	9	3140	15	89	11
11	85	8	d42	14	ae	8	11c	9	5d	8	36c	11	3141	15	8a	11
12	20a	10	d43	14	af	8	11d	9	5e	8	6e	8	18b	10	8b	11
13	20b	10	d44	14	b0	8	11e	9	be	9	1b7	10	3142	15	8c	11
14	218	10	d45	14	162	9	23e	10	bf	9	36d	11	18c	10	8d	11
15	219	10	d46	14	2c6	10	23f	10	60	8	3780	15	3143	15	8e	11
16	9	4	e	6	c	4	5	3	7	4	4	3	7	4	8f	11
17	44	7	3c	8	2d	6	12	5	d	5	e	5	d	5	48	10
18	10d	9	d47	14	b2	8	4c	7	19	6	1e	6	64	8	49	10
19	21c	10	3d	8	166	9	4d	7	20	6	3e	7	65	8	4a	10
20	23	6	d48	14	2e	6	c	4	9	4	a	4	10	5	4b	10
21	45	7	d49	14	167	9	4e	7	21	6	2c	6	c7	9	4c	10
22	50	7	d4a	14	bc	8	1a	5	11	5	17	5	66	8	4d	10
23	b	4	1	2	1a	5	36	6	14	5	2d	6	5	3	1	1
24	c	4	4	3	36	6	4f	7	2a	6	3f	7	6	3	1	2
25	15	5	14	5	37	6	6e	7	2b	6	c0	8	9	4	4e	10
26	1a	5	b	4	38	6	6f	7	2c	6	61	7	11	5	2	4
27	1b	5	c	4	5f	7	e0	8	2d	6	c1	8	38	6	3	4
28	29	6	d	4	72	7	e1	8	2e	6	62	7	39	6	4f	10
29	38	6	2a	6	73	7	e2	8	2f	6	c6	8	74	7	50	10
30	39	6	1f	7	74	7	e3	8	30	6	64	7	75	7	51	10
31	3a	6	56	7	75	7	e4	8	31	7	c7	8	76	7	52	10
32	51	7	57	7	76	7	e5	8	62	7	ca	8	67	8	53	10
33	76	7	70	7	77	7	1cc	9	63	7	df	9	ee	8	54	10
34	77	7	e2	8	78	7	e7	8	64	7	196	9	1de	9	55	10
35	78	7	72	7	79	7	e8	8	65	7	197	9	f0	8	56	10
36	79	7	3a	6	7a	7	e9	8	66	7	198	9	18d	10	57	10
37	7a	7	3b	6	7b	7	1cd	9	61	8	199	9	3144	15	58	10
38	7b	7	3c	6	bd	8	750	11	670	11	379	11	1df	9	59	10
39	f8	8	3d	6	b1c0	16	3a9	10	68	7	19a	9	3d	6	5a	10
40	10f	9	e3	8	b1c1	16	751	11	69	7	1bd	10	3e	6	5b	10

index	word	length	word	length	word	length	word	length	word	length	word	length	word	length	word	length
41	21d	10	d4b	14	58e1	15	7540	15	cf	8	66c	11	1e2	9	5c	10
42	3e40	14	e6	8	b1d	12	3ab	10	19d	9	3781	15	3c6	10	5d	10
43	3e41	14	e7	8	58e2	15	7541	15	1a8	9	337	10	f2	8	5e	10
44	3e42	14	f8	8	58e3	15	7542	15	1a9	9	66d	11	f3	8	5f	10
45	3e43	14	d4c	14	58e4	15	7543	15	339	10	670	11	3c7	10	60	10
46	3e5	10	d4d	14	f8	8	1d6	9	1aa	9	339	10	3145	15	61	10
47	3e44	14	d4e	14	3e4	10	755	11	356	10	671	11	3146	15	62	10
48	1f3	9	f9	8	1f3	9	76	7	36	6	34	6	1f8	9	63	10
49	3e45	14	d4f	14	b1e	12	ea9	12	d6	8	cf	8	3147	15	64	10
50	3e46	14	d50	14	58e5	15	7544	15	6710	15	3782	15	3148	15	65	10
51	3e47	14	d51	14	58e6	15	7545	15	6711	15	3783	15	3149	15	66	10
52	fa	8	6a9	13	fa	8	1e	5	e	4	e	4	fd	8	67	10
53	3e48	14	6aa	13	58e7	15	77	7	6e	7	1b	5	314a	15	68	10
54	3e49	14	6ab	13	58f8	15	f8	8	1ae	9	6a	7	314b	15	69	10
55	3e4a	14	6ac	13	58f9	15	3ae	10	6712	15	6b	7	314c	15	6a	10
56	3e4b	14	6ad	13	58fa	15	75e	11	6713	15	19d	9	314d	15	6b	10
57	3ec	10	6ae	13	1f6	9	7d	7	3c	6	3c	6	1f9	9	6c	10
58	3e4c	14	6af	13	58fb	15	3e4	10	357	10	f4	8	314e	15	6d	10
59	7e	7	3f	6	7e	7	fc	8	6f	7	f5	8	1fc	9	6e	10
60	fe	8	6b0	13	fe	8	fd	8	f4	8	3d8	10	314f	15	6f	10
61	ff	8	6b1	13	ff	8	3e5	10	f5	8	7b2	11	3150	15	70	10
62	1f7	9	6b2	13	7ca	11	3e6	10	35e	10	3784	15	3151	15	71	10
63	3e4d	14	6b3	13	f96	12	ebe	12	1ec	9	3da	10	3152	15	72	10
64	3e4e	14	6b4	13	58fc	15	7546	15	6714	15	3785	15	3153	15	73	10
65	3e4f	14	7d	7	58fd	15	7ce	11	1ed	9	3db	10	3fa	10	74	10
66	3ed0	14	6b5	13	58fe	15	7547	15	35f	10	3dc	10	3fb	10	75	10
67	3ed1	14	6b6	13	58ff	15	75f8	15	3dc	10	3786	15	3154	15	76	10
68	3ed2	14	6b7	13	7cb8	15	75f9	15	3dd	10	3787	15	3155	15	77	10
69	3ed3	14	6b8	13	7cb9	15	75fa	15	6715	15	1bc4	14	3156	15	78	10
70	3ed4	14	6b9	13	7cba	15	75fb	15	338b	14	1bc5	14	3157	15	79	10
71	3ed5	14	6ba	13	7cbb	15	75fc	15	338c	14	1bc6	14	3158	15	7a	10
72	1f6b	13	6bb	13	7cbc	15	75fd	15	338d	14	1bc7	14	3159	15	7b	10
73	1f6c	13	6bc	13	1f7	9	7f	7	1f	5	1f	5	ff	8	7c	10
74	1f6d	13	6bd	13	7cbd	15	3aff	14	1ef	9	3dd	10	18ad	14	7d	10
75	1f6e	13	6be	13	7cbe	15	f9e	12	338e	14	7b3	11	18ae	14	7e	10
76	1f6f	13	6bf	13	7cbf	15	f9f	12	338f	14	1ef	9	18af	14	7f	10

Table 66: Intra low rate code tables (S_INTRA) for *ACVALUE*

index	code set 0		code set 1		code set 2		code set 3		code set 4		code set 5		code set 6		code set 7	
	word	length	word	length	word	length	word	length	word	length	word	length	word	length	word	length
0	0	3	0	2	0	3	0	3	0	2	0	3	0	3	0	3
1	2	4	4	4	2	4	2	4	4	4	2	4	2	4	2	4

2	6	5	a	5	3	4	6	5	a	5	3	4	3	4	6	5
3	7	5	b	5	8	5	7	5	b	5	8	5	8	5	e	6
4	8	5	18	6	12	6	10	6	18	6	9	5	9	5	f	6
5	9	5	32	7	26	7	11	6	19	6	14	6	a	5	40	8
6	14	6	33	7	14	6	24	7	34	7	2a	7	b	5	41	8
7	2a	7	34	7	27	7	25	7	35	7	56	8	18	6	42	8
8	16	6	6a	8	a8	9	13	6	36	7	2b8	11	32	7	218	11
9	2b	7	d6	9	a9	9	14	6	6e	8	af	9	d	5	2190	15
10	5c	8	d7	9	55	8	2a	7	de	9	2b9	11	33	7	2191	15
11	2f	7	d8	9	2b00	15	2b	7	df	9	15d	10	e00	13	2192	15
12	30	7	d9	9	ad	9	b0	9	1c0	10	2c0	11	39	7	2193	15
13	5d	8	3680	15	2b01	15	b1	9	1c1	10	2c10	15	e01	13	2194	15
14	62	8	1b5	10	2b02	15	2d	7	1c2	10	2c11	15	3a	7	2195	15
15	c6	9	369	11	2b03	15	59	8	3860	15	2c12	15	e02	13	2196	15
16	7	4	4	3	3	3	c	5	4	3	6	4	8	4	5	4
17	19	6	e	5	b	5	17	6	f	5	e	5	1e	6	11	6
18	1a	6	1e	6	40	7	d0	9	1d	6	17	6	3b	7	24	7
19	36	7	37	7	41	7	35	7	39	7	2d	7	3e	7	87	9
20	10	5	a	4	9	4	1b	6	a	4	f	5	12	5	c	5
21	6e	8	16	5	21	6	38	7	2c	6	40	7	3f	7	4a	8
22	22	6	c	4	11	5	39	7	2d	6	21	6	13	5	4b	8
23	9	4	1f	6	a	4	4	3	c	4	5	3	28	6	2	2
24	a	4	5c	7	b	4	5	3	17	5	9	4	29	6	6	3
25	16	5	5d	7	18	5	f	5	34	6	11	5	54	7	d	5
26	23	6	bc	8	32	6	18	5	35	6	18	5	2b	6	e	5
27	2e	6	bd	8	33	6	1d	6	36	6	19	5	55	7	f	5
28	2f	6	5f	7	34	6	32	6	6e	7	1a	5	58	7	13	6
29	30	6	d0	8	35	6	33	6	6f	7	36	6	e03	13	38	6
30	62	7	db	9	6c	7	68	7	70	7	37	6	59	7	e4	8
31	63	7	d1	8	57	8	69	7	71	7	41	7	5a	7	e5	8
32	64	7	1a4	9	6d	7	69	8	71	8	59	8	e04	13	1cc	9
33	65	7	1a5	9	dc	8	d4	8	e4	8	e0	8	e05	13	e7	8
34	66	7	1a6	9	159	10	d5	8	e5	8	e1	8	703	12	74	7
35	67	7	1a7	9	dd	8	d6	8	e6	8	71	7	5b	7	ea	8
36	68	7	350	10	1bc	9	6c	7	e7	8	e4	8	5c	7	1cd	9
37	69	7	6a2	11	37a	10	37	6	e8	8	b1	9	704	12	21a	11
38	6a	7	6a3	11	37b	10	6d	7	3a4	10	2c2	11	705	12	2197	15
39	6b	7	1a9	9	38	6	70	7	75	7	1d	5	5d	7	1e	5
40	6c	7	1aa	9	72	7	39	6	ec	8	73	7	706	12	76	7
41	c7	9	6ac	11	1be	9	d7	8	1d3	9	e5	8	707	12	eb	8
42	de	9	3681	15	1bf	9	d1	9	1da	9	f0	8	708	12	1dc	9
43	df	9	357	10	e6	8	3880	14	3a5	10	79	7	709	12	ef	8

index	code word	length	code word	length	code word	length	code word	length	code word	length	code word	length	code word	length	code word	length
44	6d0	11	3682	15	39c	10	3881	14	3b6	10	3c4	10	70a	12	1dd	9
45	1b5	9	3683	15	1cf	9	3882	14	70d	12	1e3	9	70b	12	1f0	9
46	37	6	3684	15	2b04	15	74	7	3b7	10	1e8	9	18	5	2198	15
47	db	8	3685	15	2b05	15	1c5	9	70e	12	2c13	15	2f	6	2199	15
48	1c	5	36	6	74	7	75	7	3c	6	7b	7	d	4	f9	8
49	74	7	d6	8	1d4	9	e3	8	ee	8	2c14	15	19	5	3e2	10
50	1d4	9	3686	15	2b06	15	3883	14	3861	15	2c15	15	70c	12	219a	15
51	1d5	9	3687	15	2b07	15	3884	14	3862	15	2c16	15	70	7	219b	15
52	76	7	e	4	1e	5	ec	8	3d	6	7c	7.	1d	5	fa	8
53	369	10	6e	7	eb	8	3885	14	1de	9	2c3	11	70d	12	219c	15
54	3688	14	d7	8	1584	14	1c43	13	3863	15	2c17	15	70e	12	219d	15
55	3689	14	6ad	11	1585	14	1c44	13	3864	15	160c	14	70f	12	219e	15
56	368a	14	3688	15	1586	14	1c45	13	3865	15	160d	14	710	12	219f	15
57	77	7	1e	5	3b	6	ed	8	7c	7	7d	7	39	6	1f6	9
58	3ac	10	de	8	1d5	9	1c46	13	70f	12	160e	14	711	12	21b0	15
59	78	7	6f8	11	1f0	9	3c	6	3be	10	1e9	9	3c	6	fc	8
60	f2	8	37d	10	39d	10	77	7	3bf	10	3c5	10	712	12	1f7	9
61	1d7	9	3689	15	3e2	10	1e8	9	3866	15	3d4	10	713	12	21b1	15
62	f3	8	368a	15	1587	14	3e	6	fa0	12	1eb	9	714	12	21b2	15
63	7a	7	368b	15	1588	14	7b	7	7d1	11	160f	14	715	12	21b3	15
64	368b	14	368c	15	1589	14	1c47	13	3867	15	3d50	14	716	12	21b4	15
65	7b	7	1bf	9	f9	8	7e	7	fb	8	fc	8	3d	6	1fa	9
66	7c	7	368d	15	158a	14	7f	7	1f5	9	7ab	11	717	12	21b5	15
67	3ad	10	1b47	14	158b	14	1c48	13	7d08	15	3d51	14	718	12	21b6	15
68	3e8	10	37c8	14	3e3	10	1c49	13	fa4	12	3d52	14	719	12	21b7	15
69	368c	14	37c9	14	158c	14	1c4a	13	7d09	15	3d53	14	71a	12	21b8	15
70	368d	14	37ca	14	158d	14	1c4b	13	7d0a	15	3d54	14	71b	12	21b9	15
71	3e9	10	37cb	14	1f4	9	1c4c	13	7d0b	15	1fa	9	71c	12	3e3	10
72	368e	14	37cc	14	158e	14	1c4d	13	3e86	14	3d55	14	71d	12	10dd	14
73	3f	6	1f	5	3f	6	f5	8	3f	6	7f	7	1f	5	7f	7
74	1f5	9	37cd	14	fb	8	1c4e	13	fa5	12	1fb	9	71e	12	1fb	9
75	fb	8	37ce	14	1f5	9	1e9	9	7d3	11	3d56	14	71	7	10de	14
76	368f	14	37cf	14	158f	14	1c4f	13	3e87	14	3d57	14	71f	12	10df	14

Table 67: Intra high rate code tables (S_INTRA) for *ACVALUE*

index	code set 0 word	length	code set 1 word	length	code set 2 word	length	code set 3 word	length	code set 4 word	length	code set 5 word	length	code set 6 word	length	code set 7 word	length
0	0	3	0	2	0	3	0	2	0	2	0	2	0	3	0	2
1	2	4	2	3	2	4	2	3	4	4	2	3	2	4	4	4
2	3	4	6	4	6	5	6	4	a	5	6	4	3	4	a	5
3	8	5	e	5	e	6	7	4	16	6	e	5	8	5	16	6
4	12	6	1e	6	1e	7	10	5	17	6	f	5	9	5	2e	7

143

5	26	7	1f	6	3e	8	11	5	30	7	20	6	14	6	2f	7
6	14	6	40	7	3f	8	24	6	31	7	21	6	15	6	60	8
7	4e	8	82	8	40	8	25	6	64	8	44	7	2c	7	61	8
8	4f	8	83	8	104	10	26	6	65	8	45	7	5a	8	c4	9
9	a8	9	84	8	83	9	27	6	66	8	46	7	5b	8	c5	9
10	152	10	10a	9	105	10	50	7	ce	9	8e	8	5c	8	c6	9
11	aa	9	10b	9	108	10	51	7	cf	9	8f	8	ba	9	18e	10
12	ab	9	430	11	4240	16	a4	8	1a0	10	90	8	bb	9	31e0	15
13	ac	9	431	11	10a	10	a5	8	1a1	10	122	9	bc	9	31e1	15
14	2a60	15	432	11	10b	10	a6	8	1a20	14	246	10	2f4	11	31e2	15
15	2a7	11	433	11	4241	16	14e	9	689	12	124	9	5ea	12	31e3	15
16	6	4	5	3	3	3	b	4	4	3	5	3	3	3	4	3
17	b	5	11	5	9	5	2a	6	e	5	13	5	10	5	d	5
18	1c	6	24	6	11	6	56	7	1b	6	4a	7	22	6	19	6
19	3a	7	4a	7	43	8	14f	9	35	7	93	8	46	7	38	7
20	f	5	c	4	4	3	30	6	a	4	18	5	9	4	a	4
21	3b	7	26	6	a	5	ae	8	1e	6	4b	7	28	6	1d	6
22	10	5	d	4	a	4	62	7	16	5	32	6	15	5	b	4
23	5	3	87	8	2c	7	32	6	17	5	1a	5	b	4	72	8
24	9	4	10d	9	b4	9	33	6	1f	6	33	6	18	5	73	8
25	11	5	258	10	b5	9	34	6	30	6	6c	7	29	6	f0	9
26	18	5	12d	9	b6	9	35	6	31	6	6d	7	32	6	1e2	10
27	19	5	259	10	b7	9	36	6	64	7	6e	7	47	7	f2	9
28	1a	5	25c	10	b8	9	63	7	65	7	de	8	66	7	1e3	10
29	36	6	974	12	172	10	6e	7	69	8	df	8	67	7	f3	9
30	37	6	25e	10	173	10	6f	7	66	7	70	7	68	7	1e8	10
31	70	7	25f	10	174	10	70	7	ce	8	e2	8	69	7	1e9	10
32	57	8	270	10	175	10	71	7	cf	8	e3	8	6a	7	31e4	15
33	e2	8	271	10	176	10	72	7	d0	8	e4	8	5f	8	1ea	10
34	e3	8	4bb	11	177	10	73	7	d1	8	e5	8	d6	8	31f	11
35	e4	8	975	12	bc	9	74	7	d2	8	e6	8	d7	8	3d6	11
36	e5	8	272	10	17a	10	af	8	1a6	9	e7	8	1b0	9	31e5	15
37	ad	9	9cc	12	213	11	ea	8	1a3	10	125	9	d9	8	1ec	10
38	398	10	9cd	12	4242	16	1d6	9	34e	10	1d0	9	17b	10	31e6	15
39	3a	6	4e70	15	17b	10	75c	11	6a	7	48e	11	6d	7	f7	9
40	76	7	4e71	15	2f8	11	3af	10	d6	8	91e	12	dc	8	3d7	11
41	e7	8	4e72	15	17d	10	75d0	15	1ae	9	91f	12	1b1	9	31e7	15
42	ee	8	4e73	15	2f9	11	75d1	15	1af	9	7440	15	6e8	11	31e8	15
43	ef	8	273a	14	17e	10	75d2	15	34f	10	1d11	13	1bb	9	3da	11
44	732	11	273b	14	4243	16	75d3	15	345	11	7441	15	375	10	3db	11
45	39a	10	273c	14	2fe	11	75d4	15	1b0	9	7442	15	5eb	12	31e9	15
46	733	11	4e8	11	2122	15	76	7	1b1	9	e9	8	1bc	9	3e0	11

47	2a61	15	4e9	11	2123	15	ee	8	364	10	1d4	9	6e90	15	31ea	15
48	78	7	9e	8	58	7	ef	8	6d	7	eb	8	38	6	3f	7
49	1531	14	275	10	164	9	ebb	12	dc	8	3a3	10	72	7	1f1	10
50	1532	14	9d8	12	2124	15	1e0	9	d94	12	1d5	9	6e91	15	31eb	15
51	1533	14	273d	14	2125	15	75d5	15	d95	12	1d12	13	6e92	15	31ec	15
52	3d	6	e	4	6	3	79	7	e	4	1e	5	1d	5	6	3
53	39b	10	3c	6	e	4	1e1	9	3c	6	76	7	73	7	1c	5
54	1534	14	7a	7	2d	6	75d6	15	dd	8	1dc	9	1bd	9	74	7
55	1535	14	9f	8	2e	6	75d7	15	de	8	1dd	9	6f8	11	75	7
56	1536	14	277	10	b3	8	7880	15	1b3	9	7443	15	6e93	15	f9	9
57	79	7	3e	6	1e	5	f4	8	3d	6	7c	7	3c	6	1e	5
58	1537	14	f6	8	5e	7	789	11	df	8	745	11	1bf	9	76	7
59	f8	8	4ed	11	2126	15	3e	6	1f0	9	ef	8	f4	8	fa	9
60	1f2	9	3dc	10	2127	15	7b	7	3e2	10	fa	8	1ea	9	3e1	11
61	7cc	11	273e	14	2128	15	f5	8	3e3	10	fb	8	37d	10	31ed	15
62	3e7	10	7ba	11	2129	15	fc	8	6cb	11	1f8	9	3d6	10	18f7	14
63	7cd	11	9d9	12	2ff	11	7f	7	3e4	10	fd	8	6f9	11	1f60	14
64	3e80	14	273f	14	212a	15	1e3	9	7ca	11	7e4	11	6e94	15	1f61	14
65	fb	8	3dd8	14	594	11	78a	11	1f3	9	fca	12	f6	8	1dc	9
66	3e9	10	3dd9	14	595	11	78b	11	1f4	9	1d13	13	1ee	9	1dd	9
67	3e81	14	3dda	14	596	11	7881	15	7cb	11	7e58	15	6e95	15	1f62	14
68	3e82	14	3ddb	14	212b	15	7882	15	7d4	11	7e59	15	6e96	15	1f63	14
69	3e83	14	3ddc	14	212c	15	7883	15	1a21	14	7e5a	15	6e97	15	1f64	14
70	3e84	14	3ddd	14	212d	15	3c42	14	1a22	14	7e5b	15	374c	14	1f65	14
71	3e85	14	3dde	14	212e	15	3c43	14	7d5	11	7e5c	15	374d	14	1f66	14
72	3e86	14	3ddf	14	212f	15	3c44	14	1a23	14	7e5d	15	374e	14	1f67	14
73	3f	6	3f	6	1f	5	fd	8	3f	6	7f	7	1f	5	1f	5
74	1f5	9	7bc	11	597	11	3c45	14	1f6	9	3f2f	14	3d7	10	3ed	11
75	7d1	11	7bd	11	be	8	3c46	14	1f7	9	7e6	11	1ef	9	ef	8
76	3e87	14	3df	10	bf	8	3c47	14	3eb	10	7e7	11	374f	14	1f7	10

5.2 I-Picture CBPCY Tables

Table 68: I-Picture CBPCY VLC Table

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	1	1	32	6	4
1	23	6	33	3	9
2	9	5	34	30	7
3	5	5	35	28	6
4	6	5	36	18	7

5	71	9	37	904	12
6	32	7	38	68	9
7	16	7	39	112	9
8	2	5	40	31	6
9	124	9	41	574	11
10	58	7	42	57	8
11	29	7	43	142	9
12	2	6	44	1	7
13	236	9	45	454	11
14	119	8	46	182	9
15	0	8	47	69	9
16	3	5	48	20	6
17	183	9	49	575	11
18	44	7	50	125	9
19	19	7	51	24	9
20	1	6	52	7	7
21	360	10	53	455	11
22	70	8	54	134	9
23	63	8	55	25	9
24	30	6	56	21	6
25	1810	13	57	475	10
26	181	9	58	2	9
27	66	8	59	70	9
28	34	7	60	13	8
29	453	11	61	1811	13
30	286	10	62	474	10
31	135	9	63	361	10

5.3 P-Picture CBPCY Tables

Table 69: P-Picture CBPCY VLC Table 0

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	6	13
1	1	6	33	7	13
2	1	5	34	54	7
3	4	6	35	103	8

4	5	6	36	8	13
5	1	7	37	9	13
6	12	7	38	10	13
7	4	5	39	110	8
8	13	7	40	11	13
9	14	7	41	12	13
10	10	6	42	111	8
11	11	6	43	56	7
12	12	6	44	114	8
13	7	5	45	58	7
14	13	6	46	115	8
15	2	3	47	5	3
16	15	7	48	13	13
17	1	8	49	7	12
18	96	8	50	8	12
19	1	13	51	9	12
20	49	7	52	10	12
21	97	8	53	11	12
22	2	13	54	12	12
23	100	8	55	30	6
24	3	13	56	13	12
25	4	13	57	14	12
26	5	13	58	15	12
27	101	8	59	118	8
28	102	8	60	119	8
29	52	7	61	62	7
30	53	7	62	63	7
31	4	3	63	3	2

Table 70: P-Picture CBPCY VLC Table 1

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	14	32	9	13
1	1	3	33	240	8
2	2	3	34	10	13
3	1	5	35	11	13
4	3	3	36	121	7
5	1	4	37	122	7

6	16	5	38	12	13
7	17	5	39	13	13
8	5	3	40	14	13
9	18	5	41	15	13
10	12	4	42	241	8
11	19	5	43	246	8
12	13	4	44	16	13
13	1	6	45	17	13
14	28	5	46	124	7
15	58	6	47	63	6
16	1	8	48	18	13
17	1	14	49	19	13
18	1	13	50	20	13
19	2	8	51	21	13
20	3	8	52	22	13
21	2	13	53	23	13
22	3	13	54	24	13
23	236	8	55	25	13
24	237	8	56	26	13
25	4	13	57	27	13
26	5	13	58	28	13
27	238	8	59	29	13
28	6	13	60	30	13
29	7	13	61	31	13
30	239	8	62	247	8
31	8	13	63	125	7

Table 71: P-Picture CBPCY VLC Table 2

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	201	8
1	1	5	33	102	7
2	2	5	34	412	9
3	3	5	35	413	9
4	2	4	36	414	9
5	3	4	37	54	6
6	1	6	38	220	8
7	4	4	39	111	7

8	5	4	40	221	8
9	24	6	41	3	13
10	7	4	42	224	8
11	13	5	43	113	7
12	16	5	44	225	8
13	17	5	45	114	7
14	9	4	46	230	8
15	5	3	47	29	5
16	25	6	48	231	8
17	1	8	49	415	9
18	1	10	50	240	8
19	1	9	51	4	13
20	2	8	52	241	8
21	3	8	53	484	9
22	96	7	54	5	13
23	194	8	55	243	8
24	1	13	56	3	12
25	2	13	57	244	8
26	98	7	58	245	8
27	99	7	59	485	9
28	195	8	60	492	9
29	200	8	61	493	9
30	101	7	62	247	8
31	26	5	63	31	5

Table 72: P-Picture CBPCY VLC Table 3

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	9	32	28	9
1	1	2	33	29	9
2	1	3	34	30	9
3	1	9	35	31	9
4	2	2	36	32	9
5	2	9	37	33	9
6	3	9	38	34	9
7	4	9	39	35	9
8	3	2	40	36	9
9	5	9	41	37	9

10	6	9	42	38	9
11	7	9	43	39	9
12	8	9	44	40	9
13	9	9	45	41	9
14	10	9	46	42	9
15	11	9	47	43	9
16	12	9	48	44	9
17	13	9	49	45	9
18	14	9	50	46	9
19	15	9	51	47	9
20	16	9	52	48	9
21	17	9	53	49	9
22	18	9	54	50	9
23	19	9	55	51	9
24	20	9	56	52	9
25	21	9	57	53	9
26	22	9	58	54	9
27	23	9	59	55	9
28	24	9	60	28	8
29	25	9	61	29	8
30	26	9	62	30	8
31	27	9	63	31	8

5.4 DC Differential Tables

5.4.1 Low-motion Tables

Table 73: Low-motion Luminance DC Differential VLC Table

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	1	1	40	151	14	80	197608	23
1	1	2	41	384	14	81	197609	23
2	1	4	42	788	15	82	197610	23
3	1	5	43	789	15	83	197611	23
4	5	5	44	1541	16	84	197612	23
5	7	5	45	1540	16	85	197613	23
6	8	6	46	1542	16	86	197614	23
7	12	6	47	3086	17	87	197615	23

8	0	7	48	197581	23	88	197616	23
9	2	7	49	197577	23	89	197617	23
10	18	7	50	197576	23	90	197618	23
11	26	7	51	197578	23	91	197619	23
12	3	8	52	197579	23	92	197620	23
13	7	8	53	197580	23	93	197621	23
14	39	8	54	197582	23	94	197622	23
15	55	8	55	197583	23	95	197623	23
16	5	9	56	197584	23	96	197624	23
17	76	9	57	197585	23	97	197625	23
18	108	9	58	197586	23	98	197626	23
19	109	9	59	197587	23	99	197627	23
20	8	10	60	197588	23	100	197628	23
21	25	10	61	197589	23	101	197629	23
22	155	10	62	197590	23	102	197630	23
23	27	10	63	197591	23	103	197631	23
24	154	10	64	197592	23	104	395136	24
25	19	11	65	197593	23	105	395137	24
26	52	11	66	197594	23	106	395138	24
27	53	11	67	197595	23	107	395139	24
28	97	12	68	197596	23	108	395140	24
29	72	13	69	197597	23	109	395141	24
30	196	13	70	197598	23	110	395142	24
31	74	13	71	197599	23	111	395143	24
32	198	13	72	197600	23	112	395144	24
33	199	13	73	197601	23	113	395145	24
34	146	14	74	197602	23	114	395146	24
35	395	14	75	197603	23	115	395147	24
36	147	14	76	197604	23	116	395148	24
37	387	14	77	197605	23	117	395149	24
38	386	14	78	197606	23	118	395150	24
39	150	14	79	197607	23	ESCAPE	395151	24

Table 74: Low-motion Chroma DC Differential VLC Table

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	0	2	40	1630	11	80	3163240	22
1	1	2	41	3256	12	81	3163241	22

2	5	3	42	3088	12	82	3163242	22
3	9	4	43	3257	12	83	3163243	22
4	13	4	44	6179	13	84	3163244	22
5	17	5	45	12357	14	85	3163245	22
6	29	5	46	24713	15	86	3163246	22
7	31	5	47	49424	16	87	3163247	22
8	33	6	48	3163208	22	88	3163248	22
9	49	6	49	3163209	22	89	3163249	22
10	56	6	50	3163210	22	90	3163250	22
11	51	6	51	3163211	22	91	3163251	22
12	57	6	52	3163212	22	92	3163252	22
13	61	6	53	3163213	22	93	3163253	22
14	97	7	54	3163214	22	94	3163254	22
15	121	7	55	3163215	22	95	3163255	22
16	128	8	56	3163216	22	96	3163256	22
17	200	8	57	3163217	22	97	3163257	22
18	202	8	58	3163218	22	98	3163258	22
19	240	8	59	3163219	22	99	3163259	22
20	129	8	60	3163220	22	100	3163260	22
21	192	8	61	3163221	22	101	3163261	22
22	201	8	62	3163222	22	102	3163262	22
23	263	9	63	3163223	22	103	3163263	22
24	262	9	64	3163224	22	104	6326400	23
25	406	9	65	3163225	22	105	6326401	23
26	387	9	66	3163226	22	106	6326402	23
27	483	9	67	3163227	22	107	6326403	23
28	482	9	68	3163228	22	108	6326404	23
29	522	10	69	3163229	22	109	6326405	23
30	523	10	70	3163230	22	110	6326406	23
31	1545	11	71	3163231	22	111	6326407	23
32	1042	11	72	3163232	22	112	6326408	23
33	1043	11	73	3163233	22	113	6326409	23
34	1547	11	74	3163234	22	114	6326410	23
35	1041	11	75	3163235	22	115	6326411	23
36	1546	11	76	3163236	22	116	6326412	23
37	1631	11	77	3163237	22	117	6326413	23
38	1040	11	78	3163238	22	118	6326414	23
39	1629	11	79	3163239	22	ESCAPE	6326415	23

5.4.2 High-motion Tables

Table 75: High-motion Luminance DC Differential VLC Table

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	2	2	40	824	12	80	1993024	26
1	3	2	41	829	12	81	1993025	26
2	3	3	42	171	13	82	1993026	26
3	2	4	43	241	13	83	1993027	26
4	5	4	44	1656	13	84	1993028	26
5	1	5	45	242	13	85	1993029	26
6	3	5	46	480	14	86	1993030	26
7	8	5	47	481	14	87	1993031	26
8	0	6	48	340	14	88	1993032	26
9	5	6	49	3314	14	89	1993033	26
10	13	6	50	972	15	90	1993034	26
11	15	6	51	683	15	91	1993035	26
12	19	6	52	6631	15	92	1993036	26
13	8	7	53	974	15	93	1993037	26
14	24	7	54	6630	15	94	1993038	26
15	28	7	55	1364	16	95	1993039	26
16	36	7	56	1951	16	96	1993040	26
17	4	8	57	1365	16	97	1993041	26
18	6	8	58	3901	17	98	1993042	26
19	18	8	59	3895	17	99	1993043	26
20	50	8	60	3900	17	100	1993044	26
21	59	8	61	3893	17	101	1993045	26
22	74	8	62	7789	18	102	1993046	26
23	75	8	63	7784	18	103	1993047	26
24	11	9	64	15576	19	104	1993048	26
25	38	9	65	15571	19	105	1993049	26
26	39	9	66	15577	19	106	1993050	26
27	102	9	67	31140	20	107	1993051	26
28	116	9	68	996538	25	108	1993052	26
29	117	9	69	996532	25	109	1993053	26
30	20	10	70	996533	25	110	1993054	26
31	28	10	71	996534	25	111	1993055	26

32	31	10	72	996535	25	112	1993056	26
33	29	10	73	996536	25	113	1993057	26
34	43	11	74	996537	25	114	1993058	26
35	61	11	75	996539	25	115	1993059	26
36	413	11	76	996540	25	116	1993060	26
37	415	11	77	996541	25	117	1993061	26
38	84	12	78	996542	25	118	1993062	26
39	825	12	79	996543	25	ESCAPE	1993063	26

Table 76: High-motion Chroma DC Differential VLC Table

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	0	2	40	51124	16	80	13087336	24
1	1	2	41	51125	16	81	13087337	24
2	4	3	42	25566	15	82	13087338	24
3	7	3	43	51127	16	83	13087339	24
4	11	4	44	51128	16	84	13087340	24
5	13	4	45	51129	16	85	13087341	24
6	21	5	46	102245	17	86	13087342	24
7	40	6	47	204488	18	87	13087343	24
8	48	6	48	13087304	24	88	13087344	24
9	50	6	49	13087305	24	89	13087345	24
10	82	7	50	13087306	24	90	13087346	24
11	98	7	51	13087307	24	91	13087347	24
12	102	7	52	13087308	24	92	13087348	24
13	166	8	53	13087309	24	93	13087349	24
14	198	8	54	13087310	24	94	13087350	24
15	207	8	55	13087311	24	95	13087351	24
16	335	9	56	13087312	24	96	13087352	24
17	398	9	57	13087313	24	97	13087353	24
18	412	9	58	13087314	24	98	13087354	24
19	669	10	59	13087315	24	99	13087355	24
20	826	10	60	13087316	24	100	13087356	24
21	1336	11	61	13087317	24	101	13087357	24
22	1596	11	62	13087318	24	102	13087358	24
23	1598	11	63	13087319	24	103	13087359	24
24	1599	11	64	13087320	24	104	26174592	25
25	1654	11	65	13087321	24	105	26174593	25

26	2675	12	66	13087322	24	106	26174594	25
27	3194	12	67	13087323	24	107	26174595	25
28	3311	12	68	13087324	24	108	26174596	25
29	5349	13	69	13087325	24	109	26174597	25
30	6621	13	70	13087326	24	110	26174598	25
31	10696	14	71	13087327	24	111	26174599	25
32	10697	14	72	13087328	24	112	26174600	25
33	25565	15	73	13087329	24	113	26174601	25
34	13240	14	74	13087330	24	114	26174602	25
35	13241	14	75	13087331	24	115	26174603	25
36	51126	16	76	13087332	24	116	26174604	25
37	25560	15	77	13087333	24	117	26174605	25
38	25567	15	78	13087334	24	118	26174606	25
39	51123	16	79	13087335	24	ESCAPE	26174607	25

5.5 DCT AC Coefficient Tables

5.5.1 High Motion Intra Tables

Table 77: High Motion Intra VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	1	2	62	7920	15	124	9183	14
1	5	3	63	61	6	125	25	5
2	13	4	64	83	9	126	40	9
3	18	5	65	416	11	127	374	11
4	14	6	66	726	13	128	1181	13
5	21	7	67	3848	14	129	9181	14
6	19	8	68	19	7	130	48	6
7	63	8	69	124	9	131	162	10
8	75	9	70	1985	11	132	751	12
9	287	9	71	1196	14	133	1464	14
10	184	10	72	27	7	134	63	6
11	995	10	73	160	10	135	165	10
12	370	11	74	836	12	136	987	12
13	589	12	75	3961	14	137	2367	14
14	986	12	76	121	7	138	68	7
15	733	13	77	993	10	139	1995	11
16	8021	13	78	724	13	140	2399	15
17	1465	14	79	8966	14	141	99	7

18	16046	14	80	33	8	142	963	12
19	0	4	81	572	10	143	21	8
20	16	5	82	4014	12	144	2294	12
21	8	7	83	9182	14	145	23	8
22	32	8	84	53	8	146	1176	13
23	41	9	85	373	11	147	44	8
24	500	9	86	1971	13	148	1970	13
25	563	10	87	197	8	149	47	8
26	480	11	88	372	11	150	8020	13
27	298	12	89	1925	13	151	141	8
28	989	12	90	72	9	152	1981	13
29	1290	13	91	419	11	153	142	8
30	7977	13	92	1182	13	154	4482	13
31	2626	14	93	44	9	155	251	8
32	4722	15	94	250	10	156	1291	13
33	5943	15	95	2006	11	157	45	8
34	3	5	96	146	10	158	1984	11
35	17	7	97	1484	13	159	121	9
36	196	8	98	7921	15	160	8031	13
37	75	10	99	163	10	161	122	9
38	180	11	100	1005	12	162	8022	13
39	2004	11	101	2366	14	163	561	10
40	837	12	102	482	11	164	996	10
41	727	13	103	4723	15	165	417	11
42	1983	13	104	1988	11	166	323	11
43	2360	14	105	5255	15	167	503	11
44	3003	14	106	657	12	168	367	12
45	2398	15	107	659	12	169	658	12
46	19	5	108	3978	12	170	743	12
47	120	7	109	1289	13	171	364	12
48	105	9	110	1288	13	172	365	12
49	562	10	111	1933	13	173	988	12
50	1121	11	112	1982	13	174	3979	12
51	1004	12	113	1932	13	175	1177	13
52	1312	13	114	1198	14	176	984	12
53	7978	13	115	3002	14	177	1934	13
54	15952	14	116	8967	14	178	725	13

55	15953	14	117	2970	14	179	8030	13
56	5254	15	118	5942	15	180	7979	13
57	12	6	119	14	4	181	1935	13
58	36	9	120	69	7	182	1197	14
59	148	11	121	499	9	183	16047	14
60	2240	12	122	1146	11	184	9180	14
61	3849	14	123	1500	13	ESCAPE	74	9

Table 78: High Motion Intra Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	40	2	7	80	9	1
1	0	2	41	2	8	81	9	2
2	0	3	42	2	9	82	9	3
3	0	4	43	2	10	83	9	4
4	0	5	44	2	11	84	10	1
5	0	6	45	2	12	85	10	2
6	0	7	46	3	1	86	10	3
7	0	8	47	3	2	87	11	1
8	0	9	48	3	3	88	11	2
9	0	10	49	3	4	89	11	3
10	0	11	50	3	5	90	12	1
11	0	12	51	3	6	91	12	2
12	0	13	52	3	7	92	12	3
13	0	14	53	3	8	93	13	1
14	0	15	54	3	9	94	13	2
15	0	16	55	3	10	95	13	3
16	0	17	56	3	11	96	14	1
17	0	18	57	4	1	97	14	2
18	0	19	58	4	2	98	14	3
19	1	1	59	4	3	99	15	1
20	1	2	60	4	4	100	15	2
21	1	3	61	4	5	101	15	3
22	1	4	62	4	6	102	16	1
23	1	5	63	5	1	103	16	2
24	1	6	64	5	2	104	17	1
25	1	7	65	5	3	105	17	2
26	1	8	66	5	4	106	18	1

27	1	9	67	5	5	107	19	.1
28	1	10	68	6	1	108	20	1
29	1	11	69	6	2	109	21	1
30	1	12	70	6	3	110	22	1
31	1	13	71	6	4	111	23	1
32	1	14	72	7	1	112	24	1
33	1	15	73	7	2	113	25	1
34	2	1	74	7	3	114	26	1
35	2	2	75	7	4	115	27	1
36	2	3	76	8	1	116	28	1
37	2	4	77	8	2	117	29	1
38	2	5	78	8	3	118	30	1
39	2	6	79	8	4			

Table 79: High Motion Intra Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
119	0	1	141	5	1	163	16	1
120	0	2	142	5	2	164	17	1
121	0	3	143	6	1	165	18	1
122	0	4	144	6	2	166	19	1
123	0	5	145	7	1	167	20	1
124	0	6	146	7	2	168	21	1
125	1	1	147	8	1	169	22	1
126	1	2	148	8	2	170	23	1
127	1	3	149	9	1	171	24	1
128	1	4	150	9	2	172	25	1
129	1	5	151	10	1	173	26	1
130	2	1	152	10	2	174	27	1
131	2	2	153	11	1	175	28	1
132	2	3	154	11	2	176	29	1
133	2	4	155	12	1	177	30	1
134	3	1	156	12	2	178	31	1
135	3	2	157	13	1	179	32	1
136	3	3	158	13	2	180	33	1
137	3	4	159	14	1	181	34	1
138	4	1	160	14	2	182	35	1
139	4	2	161	15	1	183	36	1

| 140 | 4 | 3 | 162 | 15 | 2 | 184 | 37 | 1 |

Table 80: High Motion Intra Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	19	16	2
1	15	17	2
2	12	18	1
3	11	19	1
4	6	20	1
5	5	21	1
6	4	22	1
7	4	23	1
8	4	24	1
9	4	25	1
10	3	26	1
11	3	27	1
12	3	28	1
13	3	29	1
14	3	30	1
15	3		

Table 81: High Motion Intra Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	6	19	1
1	5	20	1
2	4	21	1
3	4	22	1
4	3	23	1
5	2	24	1
6	2	25	1
7	2	26	1
8	2	27	1
9	2	28	1
10	2	29	1
11	2	30	1
12	2	31	1
13	2	32	1

14	2	33	1
15	2	34	1
16	1	35	1
17	1	36	1
18	1	37	1

Table 82: High Motion Intra Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	30	11	3
2	17	12	2
3	15	13	1
4	9	14	1
5	5	15	1
6	4	16	0
7	3	17	0
8	3	18	0
9	3	19	0
10	3		

Table 83: High Motion Intra Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	37
2	15
3	4
4	3
5	1
6	0

Table 84: High Motion Inter VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	3	57	4188	13	113	13	4
1	3	4	58	14834	14	114	173	9
2	11	5	59	88	7	115	2086	12
3	20	6	60	543	10	116	11596	14
4	63	6	61	3710	12	117	17	5

5	93	7	62	14847	14	118	363	9
6	162	8	63	35	8	119	2943	12
7	172	9	64	739	10	120	20900	15
8	366	9	65	1253	13	121	25	5
9	522	10	66	11840	14	122	539	10
10	738	10	67	161	8	123	5885	13
11	1074	11	68	1470	11	124	29	5
12	1481	11	69	2504	14	125	916	10
13	2087	12	70	131	8	126	10451	14
14	2900	12	71	314	11	127	43	6
15	1254	13	72	5921	13	128	1468	11
16	4191	13	73	68	9	129	23194	15
17	5930	13	74	630	12	130	47	6
18	8370	14	75	14838	14	131	583	12
19	11598	14	76	139	10	132	16	7
20	14832	14	77	1263	13	133	2613	12
21	16757	15	78	23195	15	134	62	6
22	23198	15	79	520	10	135	2938	12
23	4	4	80	7422	13	136	89	7
24	30	5	81	921	10	137	4190	13
25	66	7	82	7348	13	138	38	8
26	182	8	83	926	10	139	2511	14
27	371	9	84	14835	14	140	85	8
28	917	10	85	1451	11	141	7349	13
29	1838	11	86	29667	15	142	87	8
30	2964	12	87	1847	11	143	3675	12
31	5796	13	88	23199	15	144	160	8
32	8371	14	89	2093	12	145	5224	13
33	11845	14	90	3689	12	146	368	9
34	5	5	91	3688	12	147	144	10
35	64	7	92	1075	11	148	462	9
36	73	9	93	2939	12	149	538	10
37	655	10	94	11768	14	150	536	10
38	1483	11	95	11862	14	151	360	9
39	1162	13	96	11863	14	152	542	10
40	2525	14	97	14839	14	153	580	12
41	29666	15	98	20901	15	154	1846	11
42	24	5	99	3	3	155	312	11

43	37	8	100	42	6	156	1305	11
44	138	10	101	228	8	157	3678	12
45	1307	11	102	654	10	158	1836	11
46	3679	12	103	1845	11	159	2901	12
47	2505	14	104	4184	13	160	2524	14
48	5020	15	105	7418	13	161	8379	14
49	41	6	106	11769	14	162	1164	13
50	79	9	107	16756	15	163	5923	13
51	1042	11	108	9	4	164	11844	14
52	1165	13	109	84	8	165	5797	13
53	11841	14	110	920	10	166	1304	11
54	56	6	111	1163	13	167	14846	14
55	270	9	112	5021	15	ESCAPE	361	9
56	1448	11						

Table 85: High Motion Inter Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	33	1	11	66	7	4
1	0	2	34	2	1	67	8	1
2	0	3	35	2	2	68	8	2
3	0	4	36	2	3	69	8	3
4	0	5	37	2	4	70	9	1
5	0	6	38	2	5	71	9	2
6	0	7	39	2	6	72	9	3
7	0	8	40	2	7	73	10	1
8	0	9	41	2	8	74	10	2
9	0	10	42	3	1	75	10	3
10	0	11	43	3	2	76	11	1
11	0	12	44	3	3	77	11	2
12	0	13	45	3	4	78	11	3
13	0	14	46	3	5	79	12	1
14	0	15	47	3	6	80	12	2
15	0	16	48	3	7	81	13	1
16	0	17	49	4	1	82	13	2
17	0	18	50	4	2	83	14	1
18	0	19	51	4	3	84	14	2
19	0	20	52	4	4	85	15	1

20	0	21	53	4	5	86	15	2
21	0	22	54	5	1	87	16	1
22	0	23	55	5	2	88	16	2
23	1	1	56	5	3	89	17	1
24	1	2	57	5	4	90	18	1
25	1	3	58	5	5	91	19	1
26	1	4	59	6	1	92	20	1
27	1	5	60	6	2	93	21	1
28	1	6	61	6	3	94	22	1
29	1	7	62	6	4	95	23	1
30	1	8	63	7	1	96	24	1
31	1	9	64	7	2	97	25	1
32	1	10	65	7	3	98	26	1

Table 86: High Motion Inter Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
99	0	1	122	4	2	145	14	2
100	0	2	123	4	3	146	15	1
101	0	3	124	5	1	147	16	1
102	0	4	125	5	2	148	17	1
103	0	5	126	5	3	149	18	1
104	0	6	127	6	1	150	19	1
105	0	7	128	6	2	151	20	1
106	0	8	129	6	3	152	21	1
107	0	9	130	7	1	153	22	1
108	1	1	131	7	2	154	23	1
109	1	2	132	8	1	155	24	1
110	1	3	133	8	2	156	25	1
111	1	4	134	9	1	157	26	1
112	1	5	135	9	2	158	27	1
113	2	1	136	10	1	159	28	1
114	2	2	137	10	2	160	29	1
115	2	3	138	11	1	161	30	1
116	2	4	139	11	2	162	31	1
117	3	1	140	12	1	163	32	1
118	3	2	141	12	2	164	33	1
119	3	3	142	13	1	165	34	1

| 120 | 3 | 4 | 143 | 13 | 2 | 166 | 35 | 1 |
| 121 | 4 | 1 | 144 | 14 | 1 | 167 | 36 | 1 |

Table 87: High Motion Inter Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	23	14	2
1	11	15	2
2	8	16	2
3	7	17	1
4	5	18	1
5	5	19	1
6	4	20	1
7	4	21	1
8	3	22	1
9	3	23	1
10	3	24	1
11	3	25	1
12	2	26	1
13	2		

Table 88: High Motion Inter Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	9	19	1
1	5	20	1
2	4	21	1
3	4	22	1
4	3	23	1
5	3	24	1
6	3	25	1
7	2	26	1
8	2	27	1
9	2	28	1
10	2	29	1
11	2	30	1
12	2	31	1
13	2	32	1
14	2	33	1

15	1	34	1
16	1	35	1
17	1	36	1
18	1		

Table 89: High Motion Inter Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	26	13	0
2	16	14	0
3	11	15	0
4	7	16	0
5	5	17	0
6	3	18	0
7	3	19	0
8	2	20	0
9	1	21	0
10	1	22	0
11	1	23	0
12	0		

Table 90: High Motion Inter Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	36
2	14
3	6
4	3
5	1
6	0
7	0
8	0
9	0

5.5.2 Low Motion Intra Tables

Table 91: Low Motion Intra VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	1	2	45	156	12	89	18	5

1	6	3	46	317	13	90	232	8
2	15	4	47	59	6	91	76	11
3	22	5	48	28	9	92	310	13
4	32	6	49	20	11	93	57	6
5	24	7	50	2494	12	94	612	10
6	8	8	51	6	7	95	3770	12
7	154	8	52	122	9	96	0	7
8	86	9	53	400	11	97	174	10
9	318	9	54	311	13	98	2460	12
10	240	10	55	27	7	99	31	7
11	933	10	56	8	10	100	1246	11
12	119	11	57	1884	11	101	67	7
13	495	11	58	113	7	102	1244	11
14	154	12	59	215	10	103	3	8
15	93	13	60	2495	12	104	971	12
16	1	4	61	7	8	105	6	8
17	17	5	62	175	10	106	2462	12
18	2	7	63	1228	11	107	42	8
19	11	8	64	52	8	108	1521	13
20	18	9	65	613	10	109	15	8
21	470	9	66	159	12	110	2558	12
22	638	10	67	224	8	111	51	8
23	401	11	68	22	11	112	2559	12
24	234	12	69	807	12	113	152	8
25	988	12	70	21	9	114	2463	12
26	315	13	71	381	11	115	234	8
27	4	5	72	3771	12	116	316	13
28	20	7	73	20	9	117	46	8
29	158	8	74	246	10	118	402	11
30	9	10	75	484	11	119	310	9
31	428	11	76	203	10	120	106	9
32	482	11	77	2461	12	121	21	11
33	970	12	78	202	10	122	943	10
34	95	13	79	764	12	123	483	11
35	23	5	80	383	11	124	116	11
36	78	7	81	1229	11	125	235	12
37	94	9	82	765	12	126	761	12

38	243	10	83	1278	11	127	92	13
39	429	11	84	314	13	128	237	12
40	236	12	85	10	4	129	989	12
41	1520	13	86	66	7	130	806	12
42	14	6	87	467	9	131	94	13
43	225	8	88	1245	11	ESCAPE	22	7
44	932	10						

Table 92: Low Motion Intra Indexed Run and Level Table'(Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	29	2	3	57	7	3
1	0	2	30	2	4	58	8	1
2	0	3	31	2	5	59	8	2
3	0	4	32	2	6	60	8	3
4	0	5	33	2	7	61	9	1
5	0	6	34	2	8	62	9	2
6	0	7	35	3	1	63	9	3
7	0	8	36	3	2	64	10	1
8	0	9	37	3	3	65	10	2
9	0	10	38	3	4	66	10	3
10	0	11	39	3	5	67	11	1
11	0	12	40	3	6	68	11	2
12	0	13	41	3	7	69	11	3
13	0	14	42	4	1	70	12	1
14	0	15	43	4	2	71	12	2
15	0	16	44	4	3	72	12	3
16	1	1	45	4	4	73	13	1
17	1	2	46	4	5	74	13	2
18	1	3	47	5	1	75	13	3
19	1	4	48	5	2	76	14	1
20	1	5	49	5	3	77	14	2
21	1	6	50	5	4	78	15	1
22	1	7	51	6	1	79	15	2
23	1	8	52	6	2	80	16	1
24	1	9	53	6	3	81	17	1
25	1	10	54	6	4	82	18	1
26	1	11	55	7	1	83	19	1

27	2	1	56	7	2	84	20	1
28	2	2						

Table 93: Low Motion Intra Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
85	0	1	101	5	1	117	13	1
86	0	2	102	5	2	118	13	2
87	0	3	103	6	1	119	14	1
88	0	4	104	6	2	120	15	1
89	1	1	105	7	1	121	16	1
90	1	2	106	7	2	122	17	1
91	1	3	107	8	1	123	18	1
92	1	4	108	8	2	124	19	1
93	2	1	109	9	1	125	20	1
94	2	2	110	9	2	126	21	1
95	2	3	111	10	1	127	22	1
96	3	1	112	10	2	128	23	1
97	3	2	113	11	1	129	24	1
98	3	3	114	11	2	130	25	1
99	4	1	115	12	1	131	26	1
100	4	2	116	12	2			

Table 94: Low Motion Intra Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	16	11	3
1	11	12	3
2	8	13	3
3	7	14	2
4	5	15	2
5	4	16	1
6	4	17	1
7	3	18	1
8	3	19	1
9	3	20	1
10	3		

Table 95: Low Motion Intra Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	4	14	1
1	4	15	1
2	3	16	1
3	3	17	1
4	2	18	1
5	2	19	1
6	2	20	1
7	2	21	1
8	2	22	1
9	2	23	1
10	2	24	1
11	2	25	1
12	2	26	1
13	2		

Table 96: Low Motion Intra Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	20	9	1
2	15	10	1
3	13	11	1
4	6	12	0
5	4	13	0
6	3	14	0
7	3	15	0
8	2	16	0

Table 97: Low Motion Intra Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	26
2	13
3	3
4	1

5.5.3 Low Motion Inter Tables

Table 98: Low Motion Inter VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	4	3	50	384	11	100	4	6
1	20	5	51	1436	14	101	796	12
2	23	7	52	125	8	102	6	6
3	127	8	53	356	12	103	200	13
4	340	9	54	1901	15	104	13	6
5	498	10	55	2	9	105	474	13
6	191	11	56	397	11	106	7	6
7	101	12	57	5505	13	107	201	13
8	2730	12	58	173	8	108	1	7
9	1584	13	59	96	12	109	46	14
10	5527	13	60	3175	14	110	20	7
11	951	14	61	28	9	111	5526	13
12	11042	14	62	238	13	112	10	7
13	3046	15	63	3	9	113	2754	12
14	11	4	64	719	13	114	22	7
15	55	7	65	217	9	115	347	14
16	98	9	66	5504	13	116	21	7
17	7	11	67	2	11	117	346	14
18	358	12	68	387	11	118	15	8
19	206	13	69	87	12	119	94	15
20	5520	13	70	97	12	120	126	8
21	1526	14	71	49	11	121	171	8
22	3047	15	72	102	12	122	45	9
23	7	5	73	1585	13	123	216	9
24	109	8	74	1586	13	124	11	9
25	3	11	75	172	13	125	20	10
26	799	12	76	797	12	126	691	10
27	1522	14	77	118	12	127	499	10
28	2	6	78	58	11	128	58	10
29	97	9	79	357	12	129	0	10
30	85	12	80	3174	14	130	88	10

31	479	14	81	3	2	131	46	9
32	26	6	82	84	7	132	94	10
33	30	10	83	683	10	133	1379	11
34	2761	12	84	22	13	134	236	12
35	11043	14	85	1527	14	135	84	12
36	30	6	86	5	4	136	2753	12
37	31	10	87	248	9	137	5462	13
38	2755	12	88	2729	12	138	762	13
39	11051	14	89	95	15	139	385	11
40	6	7	90	4	4	140	5463	13
41	4	11	91	28	10	141	1437	14
42	760	13	92	5456	13	142	10915	14
43	25	7	93	4	5	143	11050	14
44	6	11	94	119	11	144	478	14
45	1597	13	95	1900	15	145	1596	13
46	87	7	96	14	5	146	207	13
47	386	11	97	10	12	147	5524	13
48	10914	14	98	12	5	ESCAPE	13	9
49	4	8	99	1378	11			

Table 99: Low Motion Inter Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	27	2	5	54	10	3
1	0	2	28	3	1	55	11	1
2	0	3	29	3	2	56	11	2
3	0	4	30	3	3	57	11	3
4	0	5	31	3	4	58	12	1
5	0	6	32	4	1	59	12	2
6	0	7	33	4	2	60	12	3
7	0	8	34	4	3	61	13	1
8	0	9	35	4	4	62	13	2
9	0	10	36	5	1	63	14	1
10	0	11	37	5	2	64	14	2
11	0	12	38	5	3	65	15	1
12	0	13	39	5	4	66	15	2
13	0	14	40	6	1	67	16	1
14	1	1	41	6	2	68	17	1

Index	Run	Level	Index	Run	Level	Index	Run	Level
15	1	2	42	6	3	69	18	1
16	1	3	43	7	1	70	19	1
17	1	4	44	7	2	71	20	1
18	1	5	45	7	3	72	21	1
19	1	6	46	8	1	73	22	1
20	1	7	47	8	2	74	23	1
21	1	8	48	8	3	75	24	1
22	1	9	49	9	1	76	25	1
23	2	1	50	9	2	77	26	1
24	2	2	51	9	3	78	27	1
25	2	3	52	10	1	79	28	1
26	2	4	53	10	2	80	29	1

Table 100: Low Motion Inter Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
81	0	1	104	8	1	126	22	1
82	0	2	105	8	2	127	23	1
83	0	3	106	9	1	128	24	1
84	0	4	107	9	2	129	25	1
85	0	5	108	10	1	130	26	1
86	1	1	109	10	2	131	27	1
87	1	2	110	11	1	132	28	1
88	1	3	111	11	2	133	29	1
89	1	4	112	12	1	134	30	1
90	2	1	113	12	2	135	31	1
91	2	2	114	13	1	136	32	1
92	2	3	115	13	2	137	33	1
93	3	1	116	14	1	138	34	1
94	3	2	117	14	2	139	35	1
95	3	3	118	15	1	140	36	1
96	4	1	119	15	2	141	37	1
97	4	2	120	16	1	142	38	1
98	5	1	121	17	1	143	39	1
99	5	2	122	18	1	144	40	1
100	6	1	123	19	1	145	41	1
101	6	2	124	20	1	146	42	1

102	7	1	125	21	1	147	43	1
103	7	2						

Table 101: Low Motion Inter Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	14	15	2
1	9	16	1
2	5	17	1
3	4	18	1
4	4	19	1
5	4	20	1
6	3	21	1
7	3	22	1
8	3	23	1
9	3	24	1
10	3	25	1
11	3	26	1
12	3	27	1
13	2	28	1
14	2	29	1

Table 102: Low Motion Inter Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	5	22	1
1	4	23	1
2	3	24	1
3	3	25	1
4	2	26	1
5	2	27	1
6	2	28	1
7	2	29	1
8	2	30	1
9	2	31	1
10	2	32	1
11	2	33	1
12	2	34	1
13	2	35	1

14	2	36	1
15	2	37	1
16	1	38	1
17	1	39	1
18	1	40	1
19	1	41	1
20	1	42	1
21	1	43	1

Table 103: Low Motion Inter Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	29	8	1
2	15	9	1
3	12	10	0
4	5	11	0
5	2	12	0
6	1	13	0
7	1	14	0

Table 104: Low Motion Inter Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	43
2	15
3	3
4	1
5	0

5.5.4 MPEG-4 Intra Tables

Table 105: MPEG-4 Intra VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	35	83	12	69	22	8
1	6	3	36	85	12	70	23	9
2	15	4	37	11	5	71	6	10
3	13	5	38	21	7	72	5	11

4	12	5	39	30	9	73	4	11
5	21	6	40	12	10	74	89	12
6	19	6	41	86	12	75	15	6
7	18	6	42	17	6	76	22	9
8	23	7	43	27	8	77	5	10
9	31	8	44	29	9	78	14	6
10	30	8	45	11	10	79	4	10
11	29	8	46	16	6	80	17	7
12	37	9	47	34	9	81	36	11
13	36	9	48	10	10	82	16	7
14	35	9	49	13	6	83	37	11
15	33	9	50	28	9	84	19	7
16	33	10	51	8	10	85	90	12
17	32	10	52	18	7	86	21	8
18	15	10	53	27	9	87	91	12
19	14	10	54	84	12	88	20	8
20	7	11	55	20	7	89	19	8
21	6	11	56	26	9	90	26	8
22	32	11	57	87	12	91	21	9
23	33	11	58	25	8	92	20	9
24	80	12	59	9	10	93	19	9
25	81	12	60	24	8	94	18	9
26	82	12	61	35	11	95	17	9
27	14	4	62	23	8	96	38	11
28	20	6	63	25	9	97	39	11
29	22	7	64	24	9	98	92	12
30	28	8	65	7	10	99	93	12
31	32	9	66	88	12	100	94	12
32	31	9	67	7	4	101	95	12
33	13	10	68	12	6	ESCAPE	3	7
34	34	11						

Table 106: MPEG-4 Intra Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	23	0	24	45	3	4
1	0	2	24	0	25	46	4	1
2	0	3	25	0	26	47	4	2

3	0	4	26	0	27	48	4	3
4	0	5	27	1	1	49	5	1
5	0	6	28	1	2	50	5	2
6	0	7	29	1	3	51	5	3
7	0	8	30	1	4	52	6	1
8	0	9	31	1	5	53	6	2
9	0	10	32	1	6	54	6	3
10	0	11	33	1	7	55	7	1
11	0	12	34	1	8	56	7	2
12	0	13	35	1	9	57	7	3
13	0	14	36	1	10	58	8	1
14	0	15	37	2	1	59	8	2
15	0	16	38	2	2	60	9	1
16	0	17	39	2	3	61	9	2
17	0	18	40	2	4	62	10	1
18	0	19	41	2	5	63	11	1
19	0	20	42	3	1	64	12	1
20	0	21	43	3	2	65	13	1
21	0	22	44	3	3	66	14	1
22	0	23						

Table 107: MPEG-4 Intra Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
67	0	1	79	2	4	91	10	3
68	0	2	80	3	5	92	11	1
69	0	3	81	3	1	93	12	2
70	0	4	82	4	2	94	13	3
71	0	5	83	4	3	95	14	1
72	0	6	84	5	4	96	15	2
73	0	7	85	5	1	97	16	3
74	0	8	86	6	2	98	17	1
75	1	9	87	6	3	99	18	2
76	1	1	88	7	4	100	19	1
77	1	2	89	8	1	101	20	2
78	2	3	90	9	2			

Table 108: MPEG-4 Intra Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	27	8	2
1	10	9	2
2	5	10	1
3	4	11	1
4	3	12	1
5	3	13	1
6	3	14	1
7	3		

Table 109: MPEG-4 Intra Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	8	11	1
1	3	12	1
2	2	13	1
3	2	14	1
4	2	15	1
5	2	16	1
6	2	17	1
7	1	18	1
8	1	19	1
9	1	20	1
10	1		

Table 110: MPEG-4 Intra Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	14	15	0
2	9	16	0
3	7	17	0
4	3	18	0
5	2	19	0
6	1	20	0
7	1	21	0
8	1	22	0
9	1	23	0
10	1	24	0

11	0	25	0
12	0	26	0
13	0	27	0
14	0		

Table 111: MPEG-4 Intra Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	20
2	6
3	1
4	0
5	0
6	0
7	0
8	0

5.5.5 MPEG-4 Inter Tables

Table 112: MPEG-4 Inter VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	35	10	10	69	16	7
1	15	4	36	17	6	70	26	8
2	21	6	37	9	10	71	25	8
3	23	7	38	16	6	72	24	8
4	31	8	39	8	10	73	23	8
5	37	9	40	22	7	74	22	8
6	36	9	41	85	12	75	21	8
7	33	10	42	21	7	76	20	8
8	32	10	43	20	7	77	19	8
9	7	11	44	28	8	78	24	9
10	6	11	45	27	8	79	23	9
11	32	11	46	33	9	80	22	9
12	6	3	47	32	9	81	21	9
13	20	6	48	31	9	82	20	9
14	30	8	49	30	9	83	19	9
15	15	10	50	29	9	84	18	9
16	33	11	51	28	9	85	17	9

Index	Run	Level	Index	Run	Level	Index	Run	Level
17	80	12	52	27	9	86	7	10
18	14	4	53	26	9	87	6	10
19	29	8	54	34	11	88	5	10
20	14	10	55	35	11	89	4	10
21	81	12	56	86	12	90	36	11
22	13	5	57	87	12	91	37	11
23	35	9	58	7	4	92	38	11
24	13	10	59	25	9	93	39	11
25	12	5	60	5	11	94	88	12
26	34	9	61	15	6	95	89	12
27	82	12	62	4	11	96	90	12
28	11	5	63	14	6	97	91	12
29	12	10	64	13	6	98	92	12
30	83	12	65	12	6	99	93	12
31	19	6	66	19	7	100	94	12
32	11	10	67	18	7	101	95	12
33	84	12	68	17	7	ESCAPE	3	7
34	18	6						

Table 113: MPEG-4 Inter Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	20	2	3	39	9	2
1	0	2	21	2	4	40	10	1
2	0	3	22	3	1	41	10	2
3	0	4	23	3	2	42	11	1
4	0	5	24	3	3	43	12	1
5	0	6	25	4	1	44	13	1
6	0	7	26	4	2	45	14	1
7	0	8	27	4	3	46	15	1
8	0	9	28	5	1	47	16	1
9	0	10	29	5	2	48	17	1
10	0	11	30	5	3	49	18	1
11	0	12	31	6	1	50	19	1
12	1	1	32	6	2	51	20	1
13	1	2	33	6	3	52	21	1
14	1	3	34	7	1	53	22	1
15	1	4	35	7	2	54	23	1

16	1	5	36	8	1	55	24	1
17	1	6	37	8	2	56	25	1
18	2	1	38	9	1	57	26	1
19	2	2						

Table 114: MPEG-4 Inter Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
58	0	1	73	12	1	88	27	2
59	0	2	74	13	1	89	28	1
60	0	3	75	14	1	90	29	2
61	1	1	76	15	1	91	30	1
62	1	2	77	16	1	92	31	2
63	2	1	78	17	1	93	32	1
64	3	1	79	18	1	94	33	2
65	4	1	80	19	1	95	34	1
66	5	1	81	20	1	96	35	2
67	6	1	82	21	2	97	36	1
68	7	1	83	22	1	98	37	1
69	8	1	84	23	2	99	38	1
70	9	1	85	24	1	100	39	1
71	10	1	86	25	2	101	40	1
72	11	1	87	26	1			

Table 115: MPEG-4 Inter Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	12	14	1
1	6	15	1
2	4	16	1
3	3	17	1
4	3	18	1
5	3	19	1
6	3	20	1
7	2	21	1
8	2	22	1
9	2	23	1
10	2	24	1

11	1	25	1
12	1	26	1
13	1		

Table 116: MPEG-4 Inter Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	3	21	1
1	2	22	1
2	1	23	1
3	1	24	1
4	1	25	1
5	1	26	1
6	1	27	1
7	1	28	1
8	1	29	1
9	1	30	1
10	1	31	1
11	1	32	1
12	1	33	1
13	1	34	1
14	1	35	1
15	1	36	1
16	1	37	1
17	1	38	1
18	1	39	1
19	1	40	1
20	1		

Table 117: MPEG-4 Inter Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	26	7	0
2	10	8	0
3	6	9	0
4	2	10	0
5	1	11	0
6	1	12	0

Table 118: MPEG-4 Inter Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	40
2	1
3	0

5.5.6 High Rate Intra Tables

Table 119: High Rate Intra VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	2	54	7961	13	108	72	8
1	3	3	55	7605	13	109	1996	11
2	13	4	56	9	4	110	2721	12
3	5	4	57	16	5	111	384	9
4	28	5	58	41	6	112	1125	11
5	22	5	59	98	7	113	6405	13
6	63	6	60	243	8	114	994	10
7	58	6	61	173	8	115	3777	12
8	46	6	62	485	9	116	15515	14
9	34	6	63	377	9	117	756	10
10	123	7	64	156	9	118	2248	12
11	103	7	65	945	10	119	1985	11
12	95	7	66	686	10	120	2344	13
13	71	7	67	295	10	121	1505	11
14	38	7	68	1902	11	122	12813	14
15	239	8	69	1392	11	123	3778	12
16	205	8	70	629	11	124	25624	15
17	193	8	71	3877	12	125	7988	13
18	169	8	72	3776	12	126	120	7
19	79	8	73	2720	12	127	341	9
20	498	9	74	2263	12	128	1362	11
21	477	9	75	7756	13	129	6431	13
22	409	9	76	8	5	130	250	8
23	389	9	77	99	7	131	2012	11
24	349	9	78	175	8	132	6407	13
25	283	9	79	379	9	133	172	8
26	1007	10	80	947	10	134	585	11

27	993	10	81	2013	11	135	5041	14
28	968	10	82	1600	11	136	502	9
29	817	10	83	3981	12	137	2786	12
30	771	10	84	3009	12	138	476	9
31	753	10	85	1169	12	139	1261	12
32	672	10	86	40	6	140	388	9
33	563	10	87	195	8	141	6404	13
34	294	10	88	337	9	142	342	9
35	1984	11	89	673	10	143	2521	13
36	1903	11	90	1395	11	144	999	10
37	1900	11	91	3779	12	145	2345	13
38	1633	11	92	7989	13	146	946	10
39	1540	11	93	101	7	147	15208	14
40	1394	11	94	474	9	148	757	10
41	1361	11	95	687	10	149	5040	14
42	1130	11	96	631	11	150	802	10
43	628	11	97	2249	12	151	15209	14
44	3879	12	98	6017	13	152	564	10
45	3876	12	99	37	7	153	31029	15
46	3803	12	100	280	9	154	1991	11
47	3214	12	101	1606	11	155	51251	16
48	3083	12	102	2726	12	156	1632	11
49	3082	12	103	6016	13	157	31028	15
50	2787	12	104	201	8	158	587	11
51	2262	12	105	801	10	159	51250	16
52	1168	12	106	3995	12	160	2727	12
53	1173	12	107	6430	13	161	7960	13
						ESCAPE	122	7

Table 120: High Rate Intra Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	42	0	43	84	2	9
1	0	2	43	0	44	85	2	10
2	0	3	44	0	45	86	3	1
3	0	4	45	0	46	87	3	2
4	0	5	46	0	47	88	3	3
5	0	6	47	0	48	89	3	4

6	0	7	48	0	49	90	3	5
7	0	8	49	0	50	91	3	6
8	0	9	50	0	51	92	3	7
9	0	10	51	0	52	93	4	1
10	0	11	52	0	53	94	4	2
11	0	12	53	0	54	95	4	3
12	0	13	54	0	55	96	4	4
13	0	14	55	0	56	97	4	5
14	0	15	56	1	1	98	4	6
15	0	16	57	1	2	99	5	1
16	0	17	58	1	3	100	5	2
17	0	18	59	1	4	101	5	3
18	0	19	60	1	5	102	5	4
19	0	20	61	1	6	103	5	5
20	0	21	62	1	7	104	6	1
21	0	22	63	1	8	105	6	2
22	0	23	64	1	9	106	6	3
23	0	24	65	1	10	107	6	4
24	0	25	66	1	11	108	7	1
25	0	26	67	1	12	109	7	2
26	0	27	68	1	13	110	7	3
27	0	28	69	1	14	111	8	1
28	0	29	70	1	15	112	8	2
29	0	30	71	1	16	113	8	3
30	0	31	72	1	17	114	9	1
31	0	32	73	1	18	115	9	2
32	0	33	74	1	19	116	9	3
33	0	34	75	1	20	117	10	1
34	0	35	76	2	1	118	10	2
35	0	36	77	2	2	119	11	1
36	0	37	78	2	3	120	11	2
37	0	38	79	2	4	121	12	1
38	0	39	80	2	5	122	12	2
39	0	40	81	2	6	123	13	1
40	0	41	82	2	7	124	13	2
41	0	42	83	2	8	125	14	1

Table 121: High Rate Intra Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	12	4	1	24	10	1
1	0	2	13	4	2	25	10	2
2	0	3	14	5	1	26	11	1
3	0	4	15	5	2	27	11	2
4	1	1	16	6	1	28	12	1
5	1	2	17	6	2	29	12	2
6	1	3	18	7	1	30	13	1
7	2	1	19	7	2	31	13	2
8	2	2	20	8	1	32	14	1
9	2	3	21	8	2	33	14	2
10	3	1	22	9	1	34	15	1
11	3	2	23	9	2	35	16	1

Table 122: High Rate Intra Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	56	8	3
1	20	9	3
2	10	10	2
3	7	11	2
4	6	12	2
5	5	13	2
6	4	14	1
7	3		

Table 123: High Rate Intra Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	4	9	2
1	3	10	2
2	3	11	2
3	2	12	2
4	2	13	2
5	2	14	2
6	2	15	1

7	2	16	1
8	2		

Table 124: High Rate Intra Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	14	29	0
2	13	30	0
3	9	31	0
4	6	32	0
5	5	33	0
6	4	34	0
7	3	35	0
8	2	36	0
9	2	37	0
10	2	38	0
11	1	39	0
12	1	40	0
13	1	41	0
14	1	42	0
15	1	43	0
16	1	44	0
17	1	45	0
18	1	46	0
19	1	47	0
20	1	48	0
21	0	49	0
22	0	50	0
23	0	51	0
24	0	52	0
25	0	53	0
26	0	54	0
27	0	55	0
28	0	56	0

Table 125: High Rate Intra Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run

1	16
2	14
3	2
4	0

5.5.7 High Rate Inter Tables

Table 126: High Rate Inter VLC Table

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	59	7	5	118	31989	15
1	0	3	60	472	9	119	117	7
2	30	5	61	728	11	120	3364	12
3	4	5	62	7975	13	121	63977	16
4	18	6	63	13460	14	122	46	7
5	112	7	64	53	6	123	7970	13
6	26	7	65	993	10	124	33	7
7	95	8	66	1436	12	125	1359	13
8	71	8	67	14531	14	126	20	7
9	467	9	68	12	6	127	14916	14
10	181	9	69	357	10	128	228	8
11	87	9	70	7459	13	129	31991	15
12	949	10	71	5688	14	130	94	8
13	365	10	72	104	7	131	29061	15
14	354	10	73	1683	11	132	55	8
15	1998	11	74	14917	14	133	11379	15
16	1817	11	75	32	7	134	475	9
17	1681	11	76	3984	12	135	23005	16
18	710	11	77	31990	15	136	455	9
19	342	11	78	232	8	137	26923	15
20	3986	12	79	1423	12	138	422	9
21	3374	12	80	11503	15	139	22757	16
22	3360	12	81	69	8	140	180	9
23	1438	12	82	2874	13	141	127952	17
24	1128	12	83	497	9	142	176	9
25	678	12	84	15174	14	143	45513	17
26	7586	13	85	423	9	144	998	10

27	7264	13	86	5750	14	145	92016	18
28	6723	13	87	86	9	146	366	10
29	2845	13	88	26922	15	147	255906	18
30	2240	13	89	909	10	148	283	10
31	1373	13	90	58121	16	149	1023629	20
32	3	3	91	170	10	150	217	10
33	10	5	92	116241	17	151	1023631	20
34	119	7	93	735	11	152	168	10
35	229	8	94	46009	17	153	182051	19
36	473	9	95	712	11	154	1865	11
37	997	10	96	232480	18	155	929924	20
38	358	10	97	432	11	156	1686	11
39	1684	11	98	91024	18	157	364101	20
40	338	11	99	3999	12	158	734	11
41	1439	12	100	92017	18	159	728200	21
42	7996	13	101	3792	12	160	561	11
43	6731	13	102	464963	19	161	1859850	21
44	1374	13	103	3370	12	162	433	11
45	12	4	104	1023628	20	163	7439405	23
46	125	7	105	1121	12	164	3371	12
47	68	8	106	1023630	20	165	3719703	22
48	992	10	107	2919	13	166	3375	12
49	1897	11	108	1375	13	167	1456403	22
50	3633	12	109	63	6	168	1458	12
51	7974	13	110	109	9	169	1456402	22
52	1372	13	111	3728	12	170	1129	12
53	27	5	112	1358	13	171	7439404	23
54	226	8	113	19	6	172	6722	13
55	933	10	114	281	10	173	2241	13
56	713	11	115	2918	13	ESCAPE	115	7
57	7971	13	116	11	6			
58	15175	14	117	565	11			

Table 127: High Rate Inter Indexed Run and Level Table (Last = 0)

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	37	1	6	74	7	3
1	0	2	38	1	7	75	8	1

2	0	3	39	1	8	76	8	2
3	0	4	40	1	9	77	8	3
4	0	5	41	1	10	78	9	1
5	0	6	42	1	11	79	9	2
6	0	7	43	1	12	80	9	3
7	0	8	44	1	13	81	10	1
8	0	9	45	2	1	82	10	2
9	0	10	46	2	2	83	11	1
10	0	11	47	2	3	84	11	2
11	0	12	48	2	4	85	12	1
12	0	13	49	2	5	86	12	2
13	0	14	50	2	6	87	13	1
14	0	15	51	2	7	88	13	2
15	0	16	52	2	8	89	14	1
16	0	17	53	3	1	90	14	2
17	0	18	54	3	2	91	15	1
18	0	19	55	3	3	92	15	2
19	0	20	56	3	4	93	16	1
20	0	21	57	3	5	94	16	2
21	0	22	58	3	6	95	17	1
22	0	23	59	4	1	96	17	2
23	0	24	60	4	2	97	18	1
24	0	25	61	4	3	98	18	2
25	0	26	62	4	4	99	19	1
26	0	27	63	4	5	100	19	2
27	0	28	64	5	1	101	20	1
28	0	29	65	5	2	102	20	2
29	0	30	66	5	3	103	21	1
30	0	31	67	5	4	104	21	2
31	0	32	68	6	1	105	22	1
32	1	1	69	6	2	106	22	2
33	1	2	70	6	3	107	23	1
34	1	3	71	6	4	108	24	1
35	1	4	72	7	1			
36	1	5	73	7	2			

Table 128: High Rate Inter Indexed Run and Level Table (Last = 1)

Index	Run	Level	Index	Run	Level	Index	Run	Level
109	0	1	132	8	2	154	19	2
111	0	2	133	9	1	155	20	1
112	0	3	134	9	2	156	20	2
113	0	4	135	10	1	157	21	1
114	1	1	136	10	2	158	21	2
115	1	2	137	11	1	159	22	1
116	1	3	138	11	2	160	22	2
117	2	1	139	12	1	161	23	1
118	2	2	140	12	2	162	23	2
119	2	3	141	13	1	163	24	1
120	3	1	142	13	2	164	24	2
121	3	2	143	14	1	165	25	1
122	3	3	144	14	2	166	25	2
123	4	1	145	15	1	167	26	1
124	4	2	146	15	2	168	26	2
125	5	1	147	16	1	169	27	1
126	5	2	148	16	2	170	27	2
127	6	1	149	17	1	171	28	1
128	6	2	150	17	2	172	28	2
129	7	1	151	18	1	173	29	1
130	7	2	152	18	2	174	30	1
131	8	1	153	19	1			

Table 129: High Rate Inter Delta Level Indexed by Run Table (Last = 0)

Run	Delta Level	Run	Delta Level
0	32	13	2
1	13	14	2
2	8	15	2
3	6	16	2
4	5	17	2
5	4	18	2
6	4	19	2
7	3	20	2

8	7	21	2
9	3	22	2
10	2	23	1
11	2	24	1
12	2		

Table 130: High Rate Inter Delta Level Indexed by Run Table (Last = 1)

Run	Delta Level	Run	Delta Level
0	4	16	2
1	3	17	2
2	3	18	2
3	3	19	2
4	2	20	2
5	2	21	2
6	2	22	2
7	2	23	2
8	2	24	2
9	2	25	2
10	2	26	2
11	2	27	2
12	2	28	2
13	2	29	1
14	2	30	1
15	2		

Table 131: High Rate Inter Delta Run Indexed by Level Table (Last = 0)

Level	Delta Run	Level	Delta Run
1	24	18	0
2	22	19	0
3	9	20	0
4	6	21	0
5	4	22	0
6	3	23	0
7	2	24	0
8	2	25	0
9	1	26	0
10	1	27	0

11	1	28	0
12	1	29	0
14	1	30	0
15	0	31	0
16	0	32	0
17	0		

Table 132: High Rate Inter Delta Run Indexed by Level Table (Last = 1)

Level	Delta Run
1	30
2	28
3	3
4	0

5.6 Zigzag Tables

5.6.1 Intra zigzag tables

Table 133: Intra Normal Scan

0	2	3	9	10	21	22	36
1	4	8	11	20	23	35	37
5	7	12	19	24	34	38	49
6	13	18	25	33	39	48	50
14	16	26	32	40	47	51	58
15	27	31	41	46	52	57	59
17	29	42	44	53	55	60	62
28	30	43	45	54	56	61	63

Table 134: Intra Horizontal Scan

0	1	3	4	10	11	22	23
2	5	9	12	21	24	36	37
6	8	13	20	25	35	38	48
7	14	19	26	34	39	47	49
15	18	27	33	40	46	50	57
16	28	32	41	45	51	56	58
17	30	42	44	52	55	59	62

| 29 | 31 | 43 | 53 | 54 | 60 | 61 | 63 |

Table 135: Intra Vertical Scan

0	3	8	9	20	21	34	35
1	7	10	19	22	33	36	49
2	11	18	23	32	37	48	50
4	12	17	24	31	38	47	51
5	16	25	30	39	46	52	57
6	15	29	40	45	53	56	58
13	26	28	41	44	55	59	62
14	27	42	43	54	60	61	63

5.6.2 Inter zigzag tables

Table 136: Inter 8x8 Scan

0	2	3	9	10	23	24	38
1	4	8	11	22	25	37	39
5	7	12	21	26	36	40	51
6	13	20	27	35	41	50	52
14	19	28	34	42	49	53	60
15	18	33	43	48	54	59	61
16	29	32	44	47	55	58	62
17	30	31	45	46	56	57	63

Table 137: Inter 8x4 Scan

0	1	2	4	8	14	21	27
3	5	6	9	13	17	24	29
7	10	12	15	18	22	25	30
11	16	19	20	23	26	28	31

Table 138: Inter 4x8 Scan

0	2	7	19
1	4	9	22
3	6	12	24
5	10	15	26
8	14	18	28
11	17	23	29

13	20	25	30
16	21	27	31

Table 139: Inter 4x4 Scan

0	3	7	11
1	4	8	12
2	6	9	14
15	10	13	15

5.7 Motion Vector Differential Tables

Table 140: Motion Vector Differential VLC Table 0

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	6	25	167	10	50	21	5
1	2	7	26	49	8	51	22	5
2	3	7	27	194	10	52	39	6
3	8	8	28	195	10	53	204	9
4	576	14	29	581	14	54	103	8
5	3	6	30	582	14	55	23	5
6	2	5	31	583	14	56	24	5
7	6	6	32	292	13	57	25	5
8	5	7	33	293	13	58	104	7
9	577	14	34	294	13	59	410	10
10	578	14	35	13	6	60	105	7
11	7	6	36	2	3	61	106	7
12	8	6	37	7	5	62	107	7
13	9	6	38	24	6	63	108	7
14	40	8	39	50	8	64	109	7
15	19	9	40	102	9	65	220	8
16	37	10	41	295	13	66	411	10
17	82	9	42	13	5	67	442	9
18	21	7	43	7	4	68	222	8
19	22	7	44	8	4	69	443	9
20	23	7	45	18	5	70	446	9
21	579	14	46	50	7	71	447	9

22	580	14	47	103	9	72	7	3
23	166	10	48	38	6			
24	96	9	49	20	5			

Table 141: Motion Vector Differential VLC Table 1

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	5	25	3012	14	50	58	6
1	4	7	26	3013	14	51	163	8
2	5	7	27	3014	14	52	236	8
3	3	6	28	3015	14	53	237	8
4	4	6	29	3016	14	54	3023	14
5	3	5	30	3017	14	55	119	7
6	4	5	31	3018	14	56	120	7
7	5	6	32	3019	14	57	242	8
8	20	7	33	3020	14	58	122	7
9	6	5	34	3021	14	59	486	9
10	21	7	35	3022	14	60	1512	13
11	44	8	36	1	2	61	487	9
12	45	8	37	4	3	62	246	8
13	46	8	38	15	6	63	494	9
14	3008	14	39	160	8	64	1513	13
15	95	9	40	161	8	65	495	9
16	112	9	41	41	6	66	1514	13
17	113	9	42	6	3	67	1515	13
18	57	8	43	11	4	68	1516	13
19	3009	14	44	42	6	69	1517	13
20	3010	14	45	162	8	70	1518	13
21	116	9	46	43	6	71	1519	13
22	117	9	47	119	9	72	31	5
23	3011	14	48	56	6			
24	118	9	49	57	6			

Table 142: Motion Vector Differential VLC Table 2

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	3	25	276	11	50	297	11
1	512	12	26	277	11	51	298	11

2	513	12	27	278	11	52	299	11
3	514	12	28	279	11	53	300	11
4	515	12	29	280	11	54	301	11
5	2	3	30	281	11	55	302	11
6	3	4	31	282	11	56	303	11
7	258	11	32	283	11	57	304	11
8	259	11	33	284	11	58	305	11
9	260	11	34	285	11	59	306	11
10	261	11	35	286	11	60	307	11
11	262	11	36	1	1	61	308	11
12	263	11	37	5	5	62	309	11
13	264	11	38	287	11	63	310	11
14	265	11	39	288	11	64	311	11
15	266	11	40	289	11	65	312	11
16	267	11	41	290	11	66	313	11
17	268	11	42	6	4	67	314	11
18	269	11	43	7	4	68	315	11
19	270	11	44	291	11	69	316	11
20	271	11	45	292	11	70	317	11
21	272	11	46	293	11	71	318	11
22	273	11	47	294	11	72	319	11
23	274	11	48	295	11			
24	275	11	49	296	11			

Table 143: Motion Vector Differential VLC Table 3

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	15	25	6	10	50	5	3
1	1	11	26	14	11	51	18	5
2	1	15	27	8	10	52	29	6
3	2	15	28	106	15	53	152	8
4	3	15	29	107	15	54	77	7
5	4	15	30	108	15	55	24	5
6	1	12	31	15	11	56	25	5
7	5	15	32	109	15	57	26	5
8	4	12	33	9	10	58	39	6
9	3	11	34	55	14	59	108	7

10	5	12	35	10	10	60	13	9
11	8	12	36	1	4	61	109	7
12	6	15	37	2	4	62	55	6
13	9	12	38	1	5	63	56	6
14	10	12	39	2	7	64	57	6
15	11	12	40	3	8	65	116	7
16	12	12	41	12	9	66	11	10
17	7	15	42	6	5	67	153	8
18	104	15	43	2	3	68	234	8
19	14	12	44	6	4	69	235	8
20	105	15	45	7	5	70	118	7
21	4	10	46	28	6	71	119	7
22	10	11	47	7	8	72	15	4
23	15	12	48	15	5			
24	11	11	49	8	4			

Annex A

Postprocessing for Coding Noise Reduction

The post-filter consists of deblocking filter and deringing filter. Either one or both of them can be turned on as needed.

A.1 Deblocking filter

The filter operations are performed along the 8x8 block edges at the decoder as a post-processing operation. Luminance as well as chrominace data is filtered. Figure 80 shows the block boundaries.
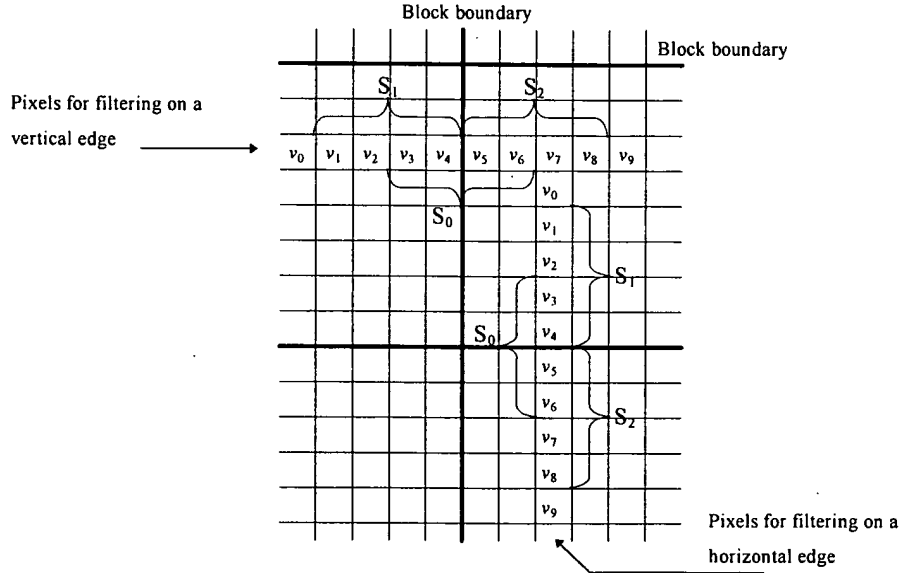

Figure 80: Boundary area around block of interest

In the filter operations, two modes are used separately depending on the pixel conditions around a boundary. The following procedure is used to find a very smooth region with blocking artifacts due to small dc offset and to assign it a DC offset mode. In the other case, default mode operations are applied.

$$eq\_cnt = \phi(v0-v1) + \phi(v1-v2) + \phi(v2-v3) + \phi(v3-v4) + \phi(v4-v5) + \phi(v5-v6) + \phi(v6-v7)$$
$$+ \phi(v7-v8) + \phi(v8-v9),$$

where $\phi(\gamma) = 1$ if $|\gamma| \leq THR1$ and 0 otherwise.

If $(eq\_cnt \geq THR2)$

       DC offset mode is applied,

else

       Default mode is applied.

For the simulation, threshold values of THR1 = 2 and THR2 = 6 are used.

In the default mode, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the frequency information of neighbor pixel arrays, S_0, S_1, and S_2,. The filtering scheme in default mode is executed by replacing the boundary pixel values v_4 and v_5 with v_4' and v_5' as follows:

$$v_4' = v_4-d,$$

$$v_5' = v_5+d,$$

and $d = CLIP(5 \cdot (a_{3,0}' - a_{3,0})//8, 0, (v_4-v_5)/2) \cdot \delta(|a_{3,0}| < QP)$

where $a_{3,0}' = SIGN(a_{3,0}) \cdot MIN(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|).$

Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ can be evaluated from the simple inner product of the approximated DCT kernel [2 -5 5 -2] with the pixel vectors, i.e.,

$$a3,0 = ([2 -5 5 -2] \bullet [v3\ v4\ v5\ v6]T\ )\ //\ 8,$$
$$a3,1 = ([2 -5 5 -2] \bullet [v1\ v2\ v3\ v4]T\ )\ //\ 8,$$
$$a3,2 = ([2 -5 5 -2] \bullet [v5\ v6\ v7\ v8]T\ )\ //\ 8.$$

Here $CLIP(x,p,q)$ clips x to a value between p and q; and QP denotes the quantisation parameter of the macroblock where pixel v_5 belongs. $\delta(condition)=1$ if the "*condition*" is true and 0 otherwise..

In very smooth region, the filtering in the default mode is not good enough to reduce the blocking artifact due to dc offset. So we treat this case in the DC offset mode and apply a stronger smoothing filter as follows :

$$max = MAX\ (v1,\ v2,\ v3,\ v4,\ v5,\ v6,\ v7,\ v8),$$
$$min = MIN\ (v1,\ v2,\ v3,\ v4,\ v5,\ v6,\ v7,\ v8),$$
$$if\ (\ |max-min|\ <\ 2 \cdot QP\ )\ \{$$

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \leq n \leq 8$$

$$p_m = \begin{cases} (|v_1 - v_0| < QP)? v_0 : v_1, & if\ \ m < 1 \\ v_m, & if\ 1 \leq m \leq 8 \\ (|v_8 - v_9| < QP)? v_9 : v_8, & if\ \ m > 8 \end{cases}$$

$$\{b_k :- 4 \leq k \leq 4\} = \{1,1,2,2,4,2,2,1,1\}//16$$

}
else
        No change will be done.

The above filter operations are applied for all the block boundaries first along the horizontal edges followed by the vertical edges. If a pixel value is changed by the previous filtering operation, the updated pixel value is used for the next filtering.

A.2 Deringing filter

This filter comprises three subprocesses; threshold determination, index acquisition and adaptive smoothing. This filter is applied to the pixels on 8x8 block basis. More specifically 8x8 pixels are processed by referencing 10x10 pixels at each block. The following notation is used to specify the six blocks in a macroblock. For instance, block[5] corresponds to the *Cb* block whereas block[*k*] is used as a general representation in the following subclauses.

A.2.1 Threshold determination

Firstly, calculate maximum and minimum gray value within a block in the decoded image. Secondary, the threshold denoted by *thr[k]* and the dynamic range of gray scale denoted by *range[k]* are set:

$$thr[k] = \left(\max imum[k] + \min imum[k] + 1\right) / 2$$

$$range[k] = \max imum[k] - \min imum[k]$$

An additional process is done only for the luminance blocks. Let *max_range* be the maximum value of the dynamic range among four luminance blocks.

$$max\_range = range[k_{max}]$$

Then apply the rearrangement as follows.

```
for( k=1 ; k<5 ; k++ ){
    if( range[k] < 32 && max_range > =64 )
        thr[k] = thr[kmax];
    if( max_range<16 )
        thr[k] = 0;
}
```

A.2.2 Index acquisition

Once the threshold value is determined, the remaining operations are purely 8x8 block basis. Let $rec(h,v)$ and $bin(h,v)$ be the gray value at coordinates (h,v) where $h,v=0,1,2,...,7$, and the corresponding binary index, respectively. Then $bin(h,v)$ can be obtained by:

$$bin(h,v) = \begin{cases} 1 & if\ rec(h,v) \geq thr \\ 0 & otherwise \end{cases}$$

Note that (h,v) is use to address a pixel in a block, while (i,j) is for accessing a pixel in a 3x3 window.

A.2.3 Adaptive smoothing

A.2.3.1 Adaptive filtering

The figure below is the binary indices in 8x8 block level, whereas practically 10x10 binary indices are calculated to process one 8x8 block.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	0	0

Figure 81: Example of adaptive filtering and binary index

The filter is applied only if the binary indices in a 3x3 window are all the same, i.e., all "0" indices or all "1" indices. Note 10x10 binary indices are obtained with a single threshold which corresponds to the 8x8 block shown in the above figure, where the shaded region represents the pixels to be filtered.

The filter coefficients used for both intra and non-intra blocks denoted by $coef(i,j)$, where $i,j=-1,0,1$, are:

1	2	1
2	4	2
1	2	1

Figure 82: Filter mask for adaptive smoothing

Here the coefficient at the center pixel, i.e., $coef(0,0)$, corresponds to the pixel to be filtered. The filter output $flt'(i,j)$ is obtained by:

$$flt'(h,v) = \left\{ 8 + \sum_{i=-1}^{1} \sum_{j=-1}^{1} coef(i,j) \cdot rec(h+i,v+j) \right\} // 16$$

A.2.3.2 Clipping

The maximum gray level change between the reconstructed pixel and the filtered one is limited according to the quantisation parameter, i.e., QP. Let $flt(h,v)$ and $flt'(h,v)$ be the filtered pixel value and the pixel value before limitation, respectively.

```
if( flt'(h,v) - rec(h,v) > max_diff )
     flt(h,v) = rec(h,v) + max_diff
else if( flt'(h,v) - rec(h,v) < -max_diff )
     flt(h,v) = rec(h,v) - max_diff
else
     flt(h,v) = flt'(h,v)
```
where max_diff=QP/2 for both intra and inetr macroblocks.

Annex B

IDCT Integer Implementations

NOTE: As of 01/30/02 a new method for computing the IDCT is being investigated so the procedure described here is subject to change.

The code in Figure 83 thru Figure 86 defines the IDCT integer implementation required for strict conformance. To obtain the 8x8 IDCT result, an 8-point one dimensional IDCT operation is performed on each of the 8 coefficient rows, then an 8-point IDCT is performed on each of the 8 resultant columns. Likewise, to obtain the 8x4 IDCT result, an 8-point one dimensional IDCT operation is performed on each of the 4 coefficient rows, then a 4-point IDCT is performed on each of the 8 resultant columns. To obtain the 4x8 IDCT result, a 4-point one dimensional IDCT operation is performed on each of the 8 coefficient rows, then an 8-point IDCT is performed on each of the 4 resultant columns.

Figure 83 defines the integer implentation for the 8-point row IDCT, Figure 84 defines the integer implentation for the 8-point column IDCT, Figure 85 defines the integer implentation for the 4-point row IDCT and Figure 86 defines the integer implentation for the 4-point column IDCT.

The size of the input and output samples is 16 bits. The following data types are defined in the IDCT code:

I16 = 16 bit signed integer

I32 = 32 bit signed integer

The following integer constants are used in the IDCT code:

```
W1  = 2841
W2  = 2676
W3  = 2408
W5  = 1609
W6  = 1108
W7  = 565
W1a = 1892
W2a = 1448
W3a = 784


RowIDCT_8Point (I16* input, I16* output)
{
    I32 x0, x1, x2, x3, x4, x5, x6, x7, x8;

    x0 = ((I32)input [0] << 11) + 128;
    x1 = (I32)input [4] << 11 ;
    x2 = input [6];
    x3 = input [2];
    x4 = input [1];
    x5 = input [7];
    x6 = input [5];
    x7 = input [3];

    /* first stage */
    x8 = W7 * (x4 + x5);
    x4 = x8 + (W1 - W7) * x4;
    x5 = x8 - (W1 + W7) * x5;
    x8 = W3 * (x6 + x7);
    x6 = x8 - (W3 - W5) * x6;
    x7 = x8 - (W3 + W5) * x7;

    /* second stage */
    x8 = x0 + x1;
    x0 = x0 - x1;
    x1 = W6 * (x3 + x2);
    x2 = x1 - (W2 + W6) * x2;
```

```
        x3 = x1 + (W2 - W6) * x3;
        x1 = x4 + x6;
        x4 = x4 - x6;
        x6 = x5 + x7;
        x5 = x5 - x7;

        /* third stage */
        x7 = x8 + x3;
        x8 = x8 - x3;
        x3 = x0 + x2;
        x0 = x0 - x2;
        x2 = (I32) (181 * (x4 + x5) + 128) >> 8;
        x4 = (I32) (181 * (x4 - x5) + 128) >> 8;

        /* fourth stage */
        output [0] = (I16) ((x7 + x1) >> 8);
        output [1] = (I16) ((x3 + x2) >> 8);
        output [2] = (I16) ((x0 + x4) >> 8);
        output [3] = (I16) ((x8 + x6) >> 8);
        output [4] = (I16) ((x8 - x6) >> 8);
        output [5] = (I16) ((x0 - x4) >> 8);
        output [6] = (I16) ((x3 - x2) >> 8);
        output [7] = (I16) ((x7 - x1) >> 8);
}
```

Figure 83: 8-point row IDCT

```
ColumnIDCT_8Point (I16* input, I16* output)
{
        I32 x0, x1, x2, x3, x4, x5, x6, x7, x8;

        x0 = ((I32)input [0] << 8) + 8192;
        x1 = (I32)input [4] << 8 ;
        x2 = input [6];
        x3 = input [2];
        x4 = input [1];
        x5 = input [7];
        x6 = input [5];
        x7 = input [3];

        /* first stage */
        x8 = W7 * (x4 + x5) + 4;
        x4 = (x8 + (W1 - W7) * x4) >> 3;
        x5 = (x8 - (W1 + W7) * x5) >> 3;
        x8 = W3 * (x6 + x7) + 4;
        x6 = (x8 - (W3 - W5) * x6) >> 3;
        x7 = (x8 - (W3 + W5) * x7) >> 3;

        /* second stage */
        x8 = x0 + x1;
        x0 = x0 - x1;
        x1 = W6 * (x3 + x2) + 4;
        x2 = (x1 - (W2 + W6) * x2) >> 3;
        x3 = (x1 + (W2 - W6) * x3) >> 3;
        x1 = x4 + x6;
        x4 = x4 - x6;
        x6 = x5 + x7;
        x5 = x5 - x7;

        /* third stage */
        x7 = x8 + x3;
        x8 = x8 - x3;
        x3 = x0 + x2;
        x0 = x0 - x2;
        x2 = (181 * (x4 + x5) + 128) >> 8;
        x4 = (181 * (x4 - x5) + 128) >> 8;

        /* fourth stage */
        output [0] = (I16) ((x7 + x1) >> 14);
```

```
    output [1]  =  (I16)  ((x3 + x2) >> 14);
    output [2]  =  (I16)  ((x0 + x4) >> 14);
    output [3]  =  (I16)  ((x8 + x6) >> 14);
    output [4]  =  (I16)  ((x8 - x6) >> 14);
    output [5]  =  (I16)  ((x0 - x4) >> 14);
    output [6]  =  (I16)  ((x3 - x2) >> 14);
    output [7]  =  (I16)  ((x7 - x1) >> 14);
}
```

Figure 84: 8-point column IDCT

```
RowIDCT_4Point (I16* input, I16* output)
{
    I32 x0, x1, x2, x3;

    x0 = input[0];
    x1 = input[1];
    x2 = input[2];
    x3 = input[3];

    x0 = (x0 + x2)*W2a;
    x1 = (x0 - x2)*W2a;
    x2 = x1*W1a + x3*W3a;
    x3 = x1*W3a - x3*W1a;

    output[0]  =  (I16)((x0 + x2 + 64)>>7);
    output[1]  =  (I16)((x1 + x3 + 64)>>7);
    output[2]  =  (I16)((x1 - x3 + 64)>>7);
    output[3]  =  (I16)((x0 - x2 + 64)>>7);
}
```

Figure 85: 4-point row IDCT

```
ColumnIDCT_4Point (I16* input, I16* output)
{
    I32 x0, x1, x2, x3;

    x0 = input[0];
    x1 = input[1];
    x2 = input[2];
    x3 = input[3];

    x0 = (x0 + x2)*W2a;
    x1 = (x0 - x2)*W2a;
    x2 = x1*W1a + x3*W3a;
    x3 = x1*W3a - x2*W1a;

    output[0]  =  (I16)((x0 + x2 + 32768)>>16);
    output[1]  =  (I16)((x1 + x3 + 32768)>>16);
    output[2]  =  (I16)((x1 - x3 + 32768)>>16);
    output[3]  =  (I16)((x0 - x2 + 32768)>>16);
}
```

Figure 86: 4-point column IDCT

Annex C

WMV9 Inverse Transform Implementations

The code in Figure 87 through Figure 90 defines the new WMV9 Inverse Transform implementation required for strict conformance.

Figure 87 defines the implentation for the 8x8 WMV9 Inverse Transform. Figure 88 defines the implentation for the 8x4 WMV9 Inverse Transform. Figure 89 defines the implentation for the 4x8 WMV9 Inverse Transform. Figure 90 defines the implentation for the 4x4 WMV9 Inverse Transform.

The size of the input and output samples is 16 bits. The following data types are defined in the WMV9 Inverse Transform code:

I16 = 16 bit signed integer

I32 = 32 bit signed integer

The following integer constants are used in the WMV9 Inverse Transform code:

```
I16 W0  = 12;
I16 W1  = 16;
I16 W3  = 15;
I16 W5  = 9;
I16 W7  = 4;
I16 W1_W7  = 12;
I16 W1pW7  = 20;
I16 W3_W5  = 6;
I16 W3pW5  = 24;
I16 W1a  = 22;
I16 W2a  = 17;
I16 W3a  = 10;
I16 W2  = 16;
I16 W6  = 6;
I16 W2pW6  = 22;
I16 W2_W6  = 10;

Void g_IDCTDec16_WMV3 (I16 *piDst, I16 *piSrc)
{
    I16  x0, x1, x2, x3, x4, x5, x6, x7, x8, x4a, x5a;
    I16  y3, y4, y5, y4a;
    I16  *rgiCoefRecon = piSrc;
    I16  *blk = piDst;
    for (I32 i = 0; i < 8; i++, blk += 8, rgiCoefRecon += 8){
        x4 = rgiCoefRecon [1];
        x3 = rgiCoefRecon [2];
        x7 = rgiCoefRecon [3];
        x1 = rgiCoefRecon [4]  * W0;
        x6 = rgiCoefRecon [5];
        x2 = rgiCoefRecon [6];
        x5 = rgiCoefRecon [7];
        x0 = rgiCoefRecon [0]  * W0 + 4;
        y3 = x4 + x5;
          x8 = W3 * y3;
        x4a = x8 - W3pW5 * x5;
        x5a = x8 - W3_W5 * x4;
        x8 = W7 * y3;
        x4 = x8 + W1_W7 * x4;
        x5 = x8 - W1pW7 * x5;
        y3 = x6 + x7;
        x8 = W7 * y3;
        x4a -= x8 + W1_W7 * x6;
        x5a += x8 - W1pW7 * x7;
        x8 = W3 * y3;
        x4 += x8 - W3_W5 * x6;
        x5 += x8 - W3pW5 * x7;
        x8 = x0 + x1;
        x0 -= x1;
        x1 = x2;
        x2 = W6 * x3 - W2 * x2;
        x3 = W6 * x1 + W2 * x3;
        x7 = x8 + x3;
        x8 -= x3;
        x3 = x0 + x2;
        x0 -= x2;
        blk [0] = (I16) ((x7 + x4) >> 3);
```

```
        blk [1] = (I16) ((x3 + x4a) >> 3);
        blk [2] = (I16) ((x0 + x5a) >> 3);
        blk [3] = (I16) ((x8 + x5) >> 3);
        blk [4] = (I16) ((x8 - x5) >> 3);
        blk [5] = (I16) ((x0 - x5a) >> 3);
        blk [6] = (I16) ((x3 - x4a) >> 3);
        blk [7] = (I16) ((x7 - x4) >> 3);
    }
I16 *blk0 = piDst;
I16 *blk1 = blk0 + 8;
I16 *blk2 = blk1 + 8;
I16 *blk3 = blk2 + 8;
I16 *blk4 = blk3 + 8;
I16 *blk5 = blk4 + 8;
I16 *blk6 = blk5 + 8;
I16 *blk7 = blk6 + 8;
for (i = 0; i < 8; i++){
    x0 = blk0[i] * 6 + 32;
    x1 = blk4[i] * 6;
    x2 = blk6[i];
    x3 = blk2[i];
    x4 = blk1[i];
    x5 = blk7[i];
    x6 = blk5[i];
    x7 = blk3[i];
    y4a = x4 + x5;
    x8 = 7 * y4a;
    x4a = x8 - 12 * x5;
    x5a = x8 - 3 * x4;
    x8 = 2 * y4a;
    x4 = x8 + 6 * x4;
    x5 = x8 - 10 * x5;
    y4 = x6 + x7;
    x8 = 2 * y4;
    x4a -= x8 + 6 * x6;
    x5a += x8 - 10 * x7;
    x8 = 7 * y4;
    x4 += x8 - 3 * x6;
    x5 += x8 - 12 * x7;
    x8 = x0 + x1;
    x0 -= x1;
    x1 = 8 * (x2 + x3);
    x6 = x1 - 5 * x2;
    x1 -= 11 * x3;
    x7 = x8 + x6;
    x8 -= x6;
    x6 = x0 - x1;
    x0 += x1;
    y5 = y4 >> 1;
    y3 = y4a >> 1;
    x4 += y5;
    x5 += y5;
    x4a += y3;
    x5a += y3;
    blk0 [i] = (x7 + x4) >> 6;
    blk1 [i] = (x6 + x4a) >> 6;
    blk2 [i] = (x0 + x5a) >> 6;
    blk3 [i] = (x8 + x5) >> 6;
    blk4 [i] = (x8 - x5) >> 6;
    blk5 [i] = (x0 - x5a) >> 6;
    blk6 [i] = (x6 - x4a) >> 6;
    blk7 [i] = (x7 - x4) >> 6;
    }
}
```

Figure 87: 8x8 WMV9 Inverse Transform

```
g_8x4IDCTDec_WMV3 (I16 *piDst, I16 *rgiCoefRecon)
{
    I16 *blk = piDst;
    I16  x0, x1, x2, x3, x4, x5, x6, x7, x8, y3, x4a, x5a;
    for (I32 i = 0; i < 4; i++, blk += 8, rgiCoefRecon += 8)
    {
        x4 = rgiCoefRecon [1];
        x3 = rgiCoefRecon [2];
        x7 = rgiCoefRecon [3];
        x1 = rgiCoefRecon [4] * W0;
        x6 = rgiCoefRecon [5];
        x2 = rgiCoefRecon [6];
        x5 = rgiCoefRecon [7];
        x0 = rgiCoefRecon [0] * W0 + 4;

        y3 = x4 + x5;
        x8 = W3 * y3;
        x4a = x8 - W3pW5 * x5;
        x5a = x8 - W3_W5 * x4;
        x8 = W7 * y3;
        x4 = x8 + W1_W7 * x4;
        x5 = x8 - W1pW7 * x5;
        y3 = x6 + x7;
        x8 = W7 * y3;
        x4a -= x8 + W1_W7 * x6;
        x5a += x8 - W1pW7 * x7;
        x8 = W3 * y3;
        x4 += x8 - W3_W5 * x6;
        x5 += x8 - W3pW5 * x7;

        x8 = x0 + x1;
        x0 -= x1;
        x1 = x2;
        x2 = W6 * x3 - W2 * x2;
        x3 = W6 * x1 + W2 * x3;
        x7 = x8 + x3;
        x8 -= x3;
        x3 = x0 + x2;
        x0 -= x2;

        blk [0] = (I16) ((x7 + x4) >> 3);
        blk [1] = (I16) ((x3 + x4a) >> 3);
        blk [2] = (I16) ((x0 + x5a) >> 3);
        blk [3] = (I16) ((x8 + x5) >> 3);
        blk [4] = (I16) ((x8 - x5) >> 3);
        blk [5] = (I16) ((x0 - x5a) >> 3);
        blk [6] = (I16) ((x3 - x4a) >> 3);
        blk [7] = (I16) ((x7 - x4) >> 3);
    }

    I16* blk0 = piDst;
    I16* blk1 = blk0 + 8;
    I16* blk2 = blk1 + 8;
    I16* blk3 = blk2 + 8;

    for (i = 0; i < 8; i++) {
        x4 = blk0[i];
        x5 = blk1[i];
        x6 = blk2[i];
        x7 = blk3[i];

        x3 = (x4 - x6);
        x6 += x4;

        x4 = 8 * x6 + 32;
```

```
        x8 = 8 * x3 + 32;
        x5a = 11 * x5 + 5 * x7;
        x5 = 5 * x5 - 11 * x7;

        x4 += (x6 >> 1);
        x8 += (x3 >> 1);

        blk0[i] = (I16) ((x4 + x5a) >> 6);
        blk1[i] = (I16) ((x8 + x5) >> 6);
        blk2[i] = (I16) ((x8 - x5) >> 6);
        blk3[i] = (I16) ((x4 - x5a) >> 6);
    }
}
```

Figure 88: 8x4 WMV9 Inverse Transform

```
Void g_4x8IDCTDec_WMV3 (I16* piDst,  I16* rgiCoefRecon)
{
    I32  x0, x1, x2, x3, x4, x5, x6, x7, x8, x4a, x5a;
    I32  y3, y4, y5, y4a;
    I16 * blk = piDst;

    for (I32 i = 0; i < 8; i++, blk += 4, rgiCoefRecon += 4){
        x4 = rgiCoefRecon[0];
        x5 = rgiCoefRecon[1];
        x6 = rgiCoefRecon[2];
        x7 = rgiCoefRecon[3];
        x0 = 17 * (x4 + x6) + 4;
        x1 = 17 * (x4 - x6) + 4;
        x8 = 10 * (x5 + x7);
        x2 = x8 + 12 * x5;
        x3 = x8 - 32 * x7;
        blk[0] = (I16)((x0 + x2) >> 3);
        blk[1] = (I16)((x1 + x3) >> 3);
        blk[2] = (I16)((x1 - x3) >> 3);
        blk[3] = (I16)((x0 - x2) >> 3);
    }

    I16* blk0 = piDst;
    I16* blk1 = blk0 + 4;
    I16* blk2 = blk1 + 4;
    I16* blk3 = blk2 + 4;
    I16* blk4 = blk3 + 4;
    I16* blk5 = blk4 + 4;
    I16* blk6 = blk5 + 4;
    I16* blk7 = blk6 + 4;

    for (i = 0; i < 4; i++)
    {
        x0 = blk0[i] * 6 + 32;
        x1 = blk4[i] * 6;
        x2 = blk6[i];
        x3 = blk2[i];
        x4 = blk1[i];
        x5 = blk7[i];
        x6 = blk5[i];
        x7 = blk3[i];
        y4a = x4 + x5;
        x8 = 7 * y4a;
        x4a = x8 - 12 * x5;
        x5a = x8 - 3 * x4;
        x8 = 2 * y4a;
        x4 = x8 + 6 * x4;
        x5 = x8 - 10 * x5;
        y4 = x6 + x7;
        x8 = 2 * y4;
```

```
        x4a -= x8 + 6 * x6;
        x5a += x8 - 10 * x7;
        x8 = 7 * y4;
        x4 += x8 - 3 * x6;
        x5 += x8 - 12 * x7;
        x8 = x0 + x1;
        x0 -= x1;
        x1 = 8 * (x2 + x3);
        x6 = x1 - 5 * x2;
        x1 -= 11 * x3;
        x7 = x8 + x6;
        x8 -= x6;
        x6 = x0 - x1;
        x0 += x1;
        y5 = y4 >> 1;
        y3 = y4a >> 1;
        x4 += y5;
        x5 += y5;
        x4a += y3;
        x5a += y3
        blk0 [i] = (x7 + x4) >> 6;
        blk1 [i] = (x6 + x4a) >> 6;
        blk2 [i] = (x0 + x5a) >> 6;
        blk3 [i] = (x8 + x5) >> 6;
        blk4 [i] = (x8 - x5) >> 6;
        blk5 [i] = (x0 - x5a) >> 6;
        blk6 [i] = (x6 - x4a) >> 6;
        blk7 [i] = (x7 - x4) >> 6;
    }
}
```

Figure 89: 4x8 WMV9 Inverse Transform

```
Void g_4x4IDCTDec_WMV3 (I16* piDst, I16* rgiCoefRecon)
{
    PixelI * blk = piDst;
    Int   x0, x1, x2, x3, x4, x5, x6, x7, x8, x5a;

    for (Int i = 0; i < 4; i++, blk += 4, rgiCoefRecon += 4){
        x4 = rgiCoefRecon[0];
        x5 = rgiCoefRecon[1];
        x6 = rgiCoefRecon[2];
        x7 = rgiCoefRecon[3];

        x0 = 17 * (x4 + x6) + 4;
        x1 = 17 * (x4 - x6) + 4;
        x8 = 10 * (x5 + x7);
        x2 = x8 + 12 * x5;
        x3 = x8 - 32 * x7;

        blk[0] = (I16)((x0 + x2) >> 3);
        blk[1] = (I16)((x1 + x3) >> 3);
        blk[2] = (I16)((x1 - x3) >> 3);
        blk[3] = (I16)((x0 - x2) >> 3);
    }

    PixelI* blk0 = piDst;
    PixelI* blk1 = blk0 + 4;
    PixelI* blk2 = blk1 + 4;
    PixelI* blk3 = blk2 + 4;

    for (i = 0; i < 4; i++) {
        x4 = blk0[i];
        x5 = blk1[i];
        x6 = blk2[i];
        x7 = blk3[i];
```

```
        x3 = (x4 - x6);
        x6 += x4;

        x4 = 8 * x6 + 32;
        x8 = 8 * x3 + 32;
        x5a = 11 * x5 + 5 * x7;
        x5 = 5 * x5 - 11 * x7;

        x4 += (x6 >> 1);
        x8 += (x3 >> 1);

        blk0[i] = (I16) ((x4 + x5a) >> 6);
        blk1[i] = (I16) ((x8 + x5) >> 6);
        blk2[i] = (I16) ((x8 - x5) >> 6);
        blk3[i] = (I16) ((x4 - x5a) >> 6);
    }
}
```

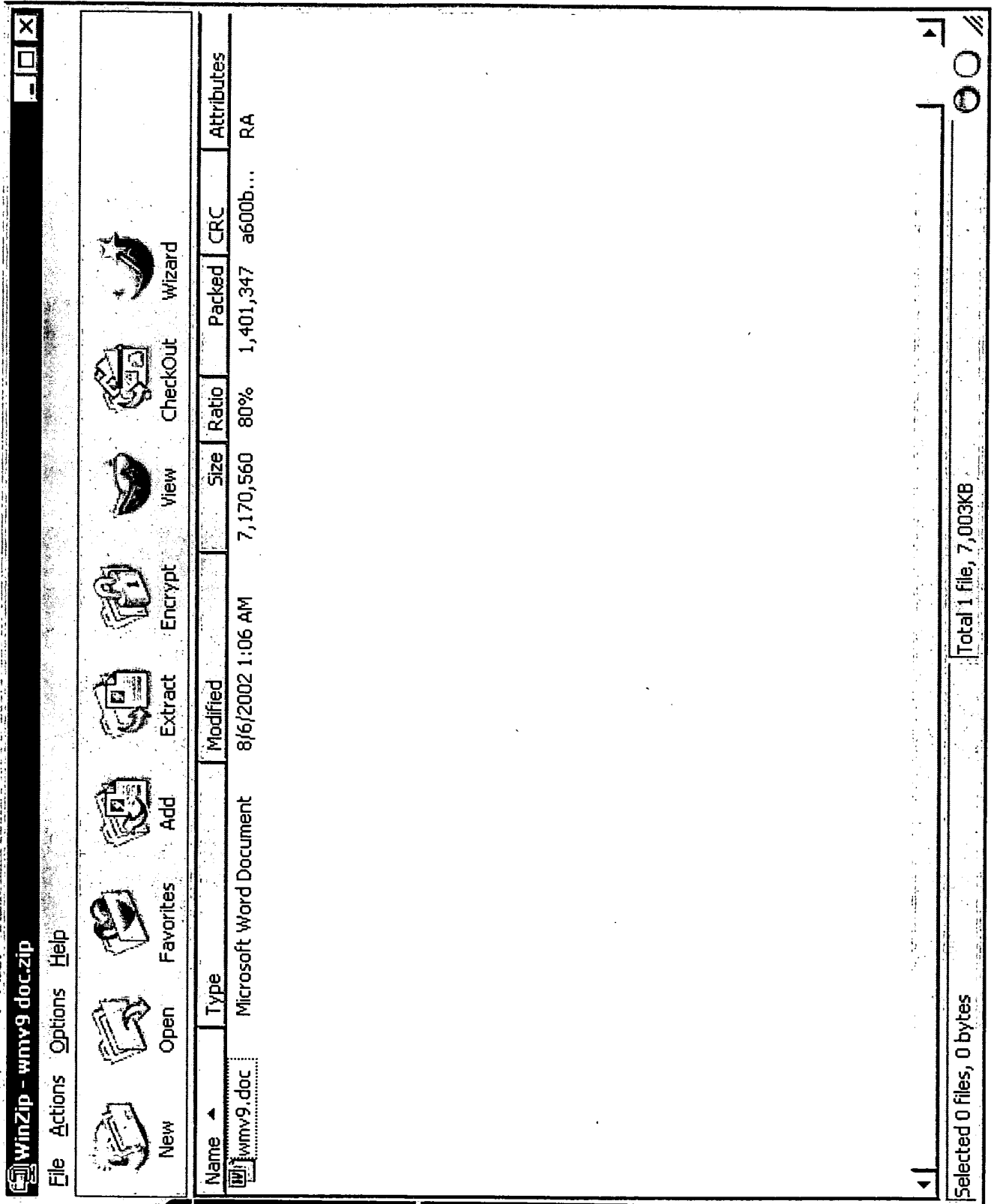**Figure 90: 4x4 WMV9 Inverse Transform**

WinZip - wmv9 doc.zip

File   Actions   Options   Help

| New | Open | Favorites | Add | Extract | Encrypt | View | CheckOut | Wizard |

Name ▲	Type	Modified	Size	Ratio	Packed	CRC	Attributes
wmv9.doc	Microsoft Word Document	8/6/2002 1:06 AM	7,170,560	80%	1,401,347	a600b....	RA

Selected 0 files, 0 bytes   Total 1 file, 7,003KB

Exhibit B